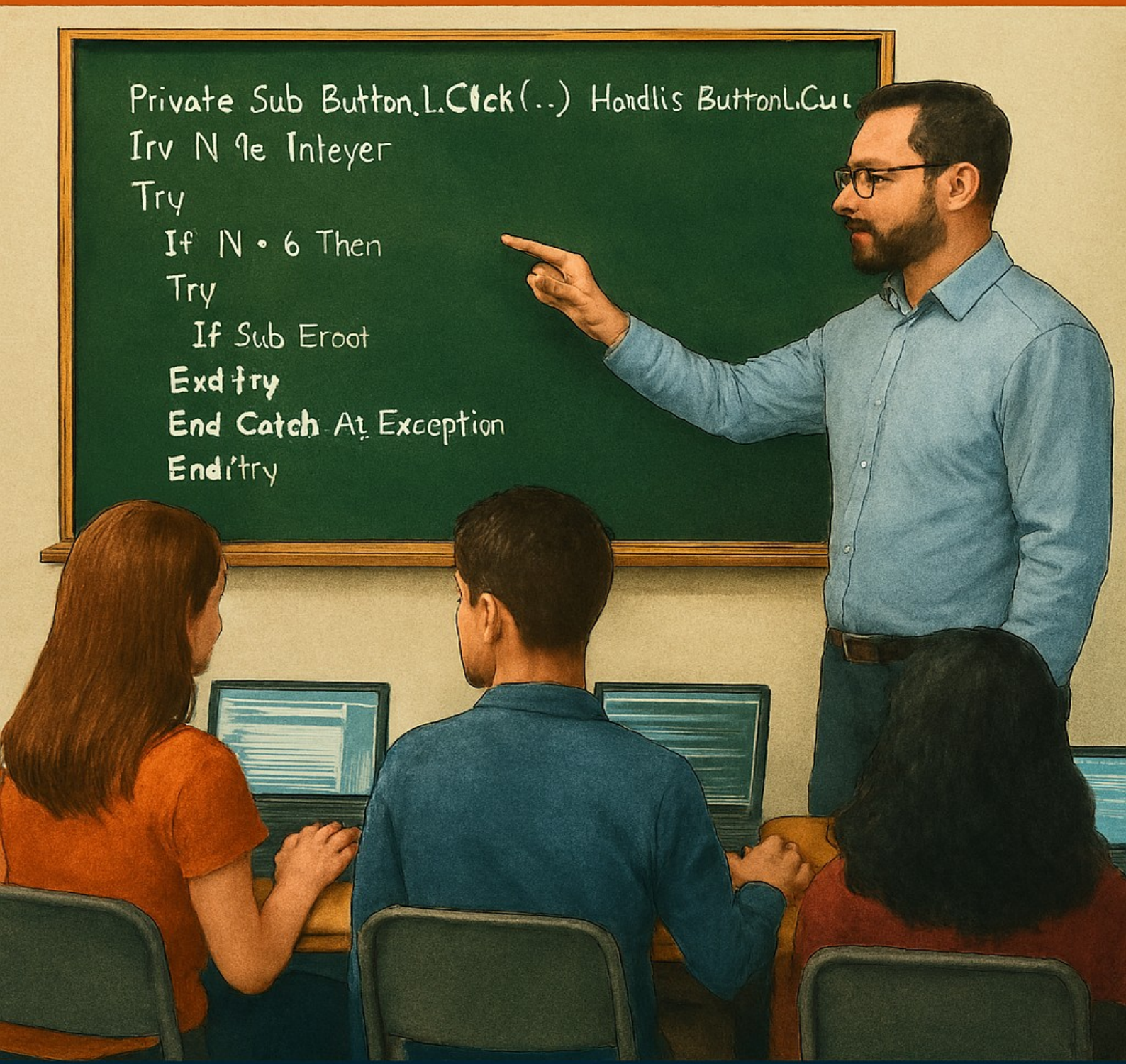
VB NET FORM

MANUAL Y EJERCICIOS v. 2



LUIS MINGUILLÓN PASCUAL

VB NET FORM

Manual y ejercicios v. 1

Luis Minguillón Pascual

COPYRIGHT

La licencia de este libro electrónico es para uso personal. Por lo tanto no puedes revenderlo a otras personas.

Gracias por respetar los derechos de autor

©Luis Minguillón Pascua
13/09/2025

Prefacio

El presente libro nace con el propósito de acompañar a todo lector en su camino de aprendizaje dentro del fascinante mundo de la programación en **Visual Basic .NET con formularios**. A partir de una primera edición con 300 ejercicios explicados, he decidido crear esta **versión ampliada**, en la cual cada problema no solo está resuelto, sino también explicado de manera más profunda y detallada, con el fin de que cada concepto, instrucción y técnica utilizada pueda ser comprendida con mayor claridad.

La programación no se trata únicamente de escribir líneas de código; es también un ejercicio de lógica, análisis y creatividad. Por ello, cada ejercicio ha sido enriquecido con explicaciones paso a paso, ejemplos prácticos y observaciones adicionales que permiten al lector no solo replicar la solución, sino **entender el porqué de cada decisión tomada en el desarrollo del programa**.

Mi mayor interés, y la razón que me ha impulsado a elaborar esta versión ampliada, es que cualquier persona, sin importar su nivel previo de conocimientos, pueda avanzar de forma más sencilla en este maravilloso mundo de la programación. Quiero que este libro sea una guía accesible, un puente entre la teoría y la práctica, y un recurso útil tanto para quienes recién comienzan como para aquellos que buscan reforzar y ampliar sus habilidades en VB.NET con formularios.

Estoy convencido de que, al comprender con mayor profundidad cada problema y cada solución, el lector podrá adquirir confianza, autonomía y la capacidad de crear sus propios proyectos. Esa es, en definitiva, la esencia de este libro: **no solo enseñar a programar, sino ayudar a pensar como programador**.

Espero que este esfuerzo contribuya a tu aprendizaje y que, al recorrer estas páginas, encuentres la motivación necesaria para continuar creciendo en este apasionante camino.

Luis Minguillón Pascual

ÍNDICE DE PROGRAMAS

- **Programa 1**. Formulario con un botón que muestra un mensaje
- **Programa 2**. Formulario con una etiqueta y un botón que cambia el texto
- **Programa 3**. Formulario con un cuadro de texto y un botón para mostrar su contenido
- **Programa 4**. Formulario con dos botones que cambian el color de fondo
- **Programa 5**. Formulario con un cuadro de texto que convierte a mayúsculas
- **Programa 6**. Formulario con dos cuadros de texto para sumar números
- **Programa 7**. Formulario con un botón para limpiar cuadros de texto
- **Programa 8**. Formulario con un botón que cierra la aplicación
- **Programa 9**. Formulario con una etiqueta que muestra la hora actual
- **Programa 10**. Formulario con un cuadro de texto que cuenta caracteres
- **Programa 11**. Formulario con dos botones para mostrar imágenes diferentes
- **Programa 12**. Formulario con un botón que cambia el tamaño del formulario
- **Programa 13**. Formulario con un cuadro de texto que solo permite números
- **Programa 14**. Formulario con tres botones que cambian el color de texto
- **Programa 15**. Formulario con un cuadro de texto y un botón para invertir texto
- **Programa 16**. Formulario con un contador de clics
- **Programa 17**. Formulario con dos botones para mostrar mensajes distintos
- **Programa 18**. Formulario con un CheckBox que habilita o deshabilita un botón
- **Programa 19**. Formulario con un TextBox multilinea y un botón para contar líneas
- **Programa 20**. Formulario con una etiqueta que cambia de color al pasar el mouse
- **Programa 21**. Formulario con un botón que cambia el ícono del formulario
- **Programa 22**. Formulario con un TrackBar que controla el tamaño del texto
- **Programa 23**. Formulario con un cuadro de texto que valida correo electrónico
- **Programa 24**. Formulario con un botón que rota imágenes
- **Programa 25**. Formulario con una lista de opciones en un ComboBox
- **Programa 26**. Formulario con un ListBox que agrega elementos desde un TextBox
- **Programa 27**. Formulario con un botón que elimina el elemento seleccionado del ListBox
- **Programa 28**. Formulario con un cuadro de texto que convierte a número romano
- **Programa 29**. Formulario con un botón que genera un número aleatorio
- **Programa 30**. Formulario con un temporizador que actualiza la hora
- **Programa 31**. Formulario con un botón que oculta y muestra una etiqueta
- **Programa 32**. Formulario con un botón que abre un nuevo formulario vacío
- **Programa 33**. Formulario con una barra de progreso que avanza al presionar un botón
- **Programa 34**. Formulario con un botón que copia texto al portapapeles
- **Programa 35**. Formulario con un cuadro de texto que limita caracteres
- **Programa 36**. Formulario con un botón que abre un cuadro de diálogo de color
- **Programa 37**. Formulario con un botón que abre un cuadro de diálogo de fuente
- **Programa 38**. Formulario con un botón que abre un cuadro de diálogo de archivo
- **Programa 39**. Formulario con un botón que abre un cuadro de diálogo de carpeta
- **Programa 40**. Formulario con un menú simple con dos opciones
- **Programa 41**. Formulario con un menú que muestra un mensaje
- **Programa 43**. Formulario con un menú para abrir otro formulario
- **Programa 44**. Formulario con una barra de herramientas (ToolStrip)

- **Programa 45**. Formulario con un cuadro de texto que busca en un ListBox
- **Programa 46**. Formulario con un botón que cambia el título del formulario
- **Programa 47**. Formulario con dos botones que mueven un Label en el formulario
- **Programa 48**. Formulario con un botón que muestra la fecha actual
- **Programa 49**. Formulario con un PictureBox que muestra diferentes imágenes
- **Programa 50**. Formulario con un botón que guarda texto en un archivo
- **Programa 51**. Agenda de contactos básica con ListBox
- **Programa 52**. Calculadora básica (suma, resta, multiplicación, división)
- **Programa 53**. Conversor de divisa (euros a dólares)
- **Programa 55**. Bloc de notas básico
- **Programa 56**. Aplicación que guarda y carga texto desde archivo
- **Programa 57**. Aplicación que cuenta palabras en un TextBox
- **Programa 58**. Aplicación que muestra tablas de multiplicar
- **Programa 59**. Aplicación que genera contraseñas aleatorias
- **Programa 60**. Aplicación que muestra mensajes aleatorios
- **Programa 61**. Aplicación que calcula el IMC
- **Programa 62**. Conversor de unidades de longitud (metros a cm, mm, km)
- **Programa 63**. Conversor de unidades de peso (kg a g, lb)
- **Programa 64**. Conversor de unidades de tiempo (horas a minutos y segundos)
- **Programa 65**. Aplicación que guarda configuración de usuario
- **Programa 67**. Calculadora de área de figuras geométricas (círculo y cuadrado)
- **Programa 68**. Calculadora de perímetro de figuras geométricas (círculo y cuadrado)
- **Programa 69**. Juego del número secreto
- **Programa 70**. Juego de adivinar palabra
- **Programa 71**. Juego de piedra, papel o tijera
- **Programa 72**. Cronómetro con Start, Stop y Reset
- **Programa 73**. Temporizador con alarma
- **Programa 74**. Editor de listas (agregar, modificar y borrar)
- **Programa 75**. Editor de texto con guardar y abrir archivo
- **Programa 76**. Aplicación con varios formularios enlazados
- **Programa 77**. Agenda que guarda datos en archivo de texto
- **Programa 78**. Agenda que guarda datos en archivo CSV
- **Programa 79**. Agenda que guarda datos en archivo XML
- **Programa 80**. Agenda que guarda datos en base de datos Access
- **Programa 81**. Formulario con gráficos simples
- **Programa 82**. Formulario con impresión de texto
- **Programa 83**. Calculadora de ecuaciones de segundo grado
- **Programa 84**. Calculadora de interés simple
- **Programa 85**. Calculadora de interés compuesto
- **Programa 86**. Conversor de números decimales a binarios
- **Programa 87**. Conversor de números binarios a decimales
- **Programa 88**. Conversor de números romanos a enteros
- **Programa 89**. Conversor de horas a minutos y segundos
- **Programa 90**. Conversor de kilogramos a libras
- **Programa 91**. Conversor de metros a pies
- **Programa 92**. Conversor de temperaturas (Celsius a Fahrenheit y Kelvin)

- **Programa 93**. Conversor de binario a hexadecimal
- **Programa 94**. Conversor de hexadecimal a decimal
- **Programa 95**. Juego de memoria con botones
- **Programa 96**. Juego de laberinto con teclas de dirección
- **Programa 97**. Juego de dados
- **Programa 98**. Juego de ahorcado
- **Programa 99**. Juego de tic-tac-toe
- **Programa 100**. Juego de cartas simple
- **Programa 101**. Juego de lotería con números aleatorios
- **Programa 102**. Juego de ruleta simple
- **Programa 103**. Juego de preguntas con respuesta simple
- **Programa 104**. Juego de carrera de carros
- **Programa 105**. Juego de tiro al blanco
- **Programa 106**. Juego de laberinto con teclas de dirección
- **Programa 107**. Agenda con búsqueda por nombre
- **Programa 108**. Agenda con edición de contactos
- **Programa 109**. Agenda con borrado de contactos
- **Programa 110**. Agenda con exportación a archivo de texto
- **Programa 111**. Agenda con importación desde archivo de texto
- **Programa 112**. Agenda con exportación a Excel
- **Programa 113**. Agenda con importación desde Excel
- **Programa 114**. Agenda con base de datos SQLite
- **Programa 115**. Agenda con base de datos SQL Server
- **Programa 116**. Agenda con base de datos MySQL
- **Programa 117**. Agenda con base de datos PostgreSQL
- **Programa 118**. Aplicación de notas con recordatorio
- **Programa 119**. Selector de fechas con MonthCalendar
- **Programa 120**. Agenda con calendario integrado
- **Programa 121**. Agenda con alarma
- **Programa 122**. Agenda con notificaciones
- **Programa 123**. Agenda con múltiples formularios
- **Programa 124**. Agenda con login de usuario
- **Programa 125**. Agenda con niveles de usuario
- **Programa 126**. Agenda con encriptación de datos
- **Programa 127**. Agenda con desencriptación de datos
- **Programa 128**. Calculadora científica simple
- **Programa 129**. Calculadora con funciones trigonométricas
- **Programa 130**. Calculadora con funciones logarítmicas
- **Programa 131**. Calculadora con potencias y raíces
- **Programa 132**. Calculadora con historial de operaciones
- **Programa 133**. Calculadora con memoria de resultados
- **Programa 134**. Juego de preguntas con puntuación
- **Programa 135**. Juego de preguntas con tiempo límite
- **Programa 136**. Juego de preguntas con niveles de dificultad
- **Programa 137**. Juego de preguntas con base de datos
- **Programa 138**. Juego de preguntas con imágenes

- **Programa 139**. Juego de preguntas con audio
- **Programa 140**. Juego de preguntas multijugador
- **Programa 141**. Juego de preguntas con ranking
- **Programa 142**. Juego de preguntas con exportación de resultados
- **Programa 143**. Juego de preguntas con importación de resultados
- **Programa 144**. Juego de preguntas con estadísticas
- **Programa 145**. Juego de preguntas con gráficos
- **Programa 146**. Juego de preguntas con personalización
- **Programa 147**. Juego de preguntas con guardado automático
- **Programa 148**. Juego de preguntas con recuperación de partida
- **Programa 149**. Juego de preguntas con dificultad progresiva
- **Programa 150**. Juego de preguntas con logros desbloqueables
- **Programa 151**. Aplicación de filtro de imágenes (escala de grises)
- **Programa 152**. Reproductor de audio WAV básico
- **Programa 153**. Reproductor de música básico
- **Programa 154**. Reproductor de video básico
- **Programa 155**. Reproductor multimedia integrado
- **Programa 156**. Aplicación que controla volumen del sistema
- **Programa 157**. Aplicación que controla brillo de pantalla
- **Programa 158**. Aplicación que toma capturas de pantalla
- **Programa 159**. Aplicación que graba sonido
- **Programa 160**. Aplicación que graba video (Webcam)
- **Programa 161**. Aplicación que reproduce listas de reproducción
- **Programa 162**. Aplicación que descarga archivos de internet
- **Programa 163**. Aplicación que sube archivos a FTP
- **Programa 164**. Cliente de servicio web SOAP
- **Programa 165**. Cliente de servicio web REST JSON
- **Programa 166**. Cliente de servicio web REST XML
- **Programa 168**. Cliente de chat local simple
- **Programa 169**. Cliente de chat en red local
- **Programa 170**. Cliente de chat con base de datos
- **Programa 171**. Cliente de chat con login de usuario
- **Programa 172**. Cliente de chat con historial de mensajes
- **Programa 173**. Cliente de chat con emoticonos
- **Programa 174**. Cliente de chat con archivos adjuntos
- **Programa 175**. Cliente de chat con múltiples salas
- **Programa 176**. Cliente de chat con videollamadas
- **Programa 177**. Cliente de chat con llamadas de voz
- **Programa 178**. Cliente de chat con notificaciones
- **Programa 179**. Cliente de chat con moderadores
- **Programa 180**. Cliente de chat con administración
- **Programa 181**. Cliente de chat con cifrado
- **Programa 182**. Cliente de chat con recuperación de historial
- **Programa 183**. Cliente de chat con estadísticas
- **Programa 184**. Cliente de chat con exportación de mensajes
- **Programa 185**. Cliente de chat con importación de mensajes

- **Programa 186**. Cliente de correo electrónico básico
- **Programa 187**. Cliente de correo con envío SMTP
- **Programa 188**. Cliente de correo con recepción POP3
- **Programa 189**. Cliente de correo con conexión IMAP
- **Programa 190**. Cliente de correo con adjuntos
- **Programa 191**. Juego de preguntas con ranking
- **Programa 192**. Juego de preguntas con exportación de resultados
- **Programa 193**. Juego de preguntas con importación de resultados
- **Programa 194**. Juego de preguntas con estadísticas
- **Programa 195**. Juego de preguntas con gráficos
- **Programa 196**. Juego de preguntas con personalización
- **Programa 197**. Juego de preguntas con guardado automático
- **Programa 198**. Juego de preguntas con recuperación de partida
- **Programa 199**. Juego de preguntas con dificultad progresiva
- **Programa 200**. Cliente de correo con múltiples cuentas
- **Programa 201**. Cliente de correo con filtros de mensajes
- **Programa 202**. Cliente de correo con carpetas personalizadas
- **Programa 203**. Cliente de correo con notificaciones
- **Programa 204**. Cliente de correo con contactos
- **Programa 205**. Cliente de correo con calendario
- **Programa 206**. Cliente de correo con recordatorios
- **Programa 207**. Cliente de correo con sincronización
- **Programa 208**. Cliente de correo con cifrado
- **Programa 209**. Cliente de correo con firmas personalizadas
- **Programa 210**. Sistema de login con logs de actividad
- **Programa 211**. Sistema de login con auditoría
- **Programa 212**. Sistema de login con autenticación externa
- **Programa 213**. Sistema de login con validación de IP
- **Programa 214**. Sistema de login con tiempo de sesión
- **Programa 215**. Sistema de login con recordatorio de sesión
- **Programa 216**. Sistema de login con registro de actividad
- **Programa 217**. Sistema de login con acceso remoto
- **Programa 218**. Sistema de login con sincronización de datos
- **Programa 219**. Sistema de login con cifrado de base de datos
- **Programa 220**. Sistema de login con hash de contraseñas
- **Programa 221**. Sistema de inventario básico
- **Programa 222**. Sistema de inventario con base de datos
- **Programa 223**. Sistema de inventario con facturación
- **Programa 224**. Sistema de inventario con reportes
- **Programa 225**. Sistema de inventario con gráficos
- **Programa 226**. Sistema de inventario con exportación Excel
- **Programa 227**. Sistema de inventario con importación Excel
- **Programa 228**. Sistema de inventario con códigos de barra
- **Programa 229**. Sistema de inventario con múltiples usuarios
- **Programa 230**. Sistema de inventario con permisos
- **Programa 231**. Sistema de inventario con auditoría

- **Programa 232**. Sistema de inventario con estadísticas
- **Programa 233**. Sistema de inventario con gráficos dinámicos
- **Programa 234**. Sistema de inventario con múltiples almacenes
- **Programa 235**. Sistema de inventario con conexión remota
- **Programa 236**. Sistema de inventario con backup automático
- **Programa 237**. Sistema de inventario con restauración de datos
- **Programa 238**. Sistema de inventario con notificaciones
- **Programa 239**. Sistema de inventario con control de lotes
- **Programa 240**. Sistema de inventario con caducidades
- **Programa 241**. Sistema de ventas básico
- **Programa 242**. Sistema de ventas con base de datos
- **Programa 243**. Sistema de ventas con clientes
- **Programa 244**. Sistema de ventas con facturación
- **Programa 245**. Sistema de ventas con reportes
- **Programa 246**. Sistema de ventas con estadísticas
- **Programa 247**. Sistema de ventas con gráficos
- **Programa 248**. Sistema de ventas con exportación Excel
- **Programa 249**. Sistema de ventas con importación Excel
- **Programa 250**. Sistema de ventas con múltiples usuarios
- **Programa 251**. Sistema de ventas con permisos
- **Programa 252**. Sistema de ventas con auditoría
- **Programa 253**. Sistema de ventas con conexión remota
- **Programa 254**. Sistema de ventas con backup automático
- **Programa 255**. Sistema de ventas con restauración de datos
- **Programa 256**. Sistema de ventas con notificaciones
- **Programa 257**. Sistema de ventas con control de stock
- **Programa 258**. Sistema de ventas con caducidades
- **Programa 259**. Sistema de ventas con código de barras
- **Programa 260**. Sistema de ventas con descuentos
- **Programa 261**. Sistema escolar básico
- **Programa 262**. Sistema escolar con base de datos
- **Programa 263**. Sistema escolar con registro de alumnos
- **Programa 264**. Sistema escolar con registro de profesores
- **Programa 265**. Sistema escolar con registro de materias
- **Programa 266**. Sistema escolar con registro de calificaciones
- **Programa 267**. Sistema escolar con reportes
- **Programa 268**. Sistema escolar con gráficos
- **Programa 269**. Sistema escolar con exportación Excel
- **Programa 270**. Sistema escolar con importación Excel
- **Programa 271**. Sistema escolar con múltiples usuarios
- **Programa 272**. Sistema escolar con permisos
- **Programa 273**. Sistema escolar con auditoría
- **Programa 274**. Sistema escolar con conexión remota
- **Programa 275**. Sistema escolar con backup automático
- **Programa 276**. Sistema escolar con restauración de datos
- **Programa 277**. Sistema escolar con notificaciones

- **Programa 278**. Sistema escolar con estadísticas
- **Programa 279**. Sistema escolar con boletines
- **Programa 280**. Sistema escolar con login de usuarios
- **Programa 281**. Sistema médico básico
- **Programa 282**. Sistema médico con registro de pacientes
- **Programa 283**. Sistema médico con registro de doctores
- **Programa 284**. Sistema médico con registro de citas
- **Programa 285**. Sistema médico con historial clínico
- **Programa 286**. Sistema médico con facturación
- **Programa 287**. Sistema médico con reportes
- **Programa 288**. Sistema médico con gráficos
- **Programa 289**. Sistema médico con exportación Excel
- **Programa 290**. Sistema médico con importación Excel
- **Programa 291**. Sistema médico con múltiples usuarios
- **Programa 292**. Sistema médico con permisos
- **Programa 293**. Sistema médico con auditoría
- **Programa 294**. Sistema médico con conexión remota
- **Programa 295**. Sistema médico con backup automático
- **Programa 296**. Sistema médico con restauración de datos
- **Programa 297**. Sistema médico con notificaciones
- **Programa 298**. Sistema médico con recordatorios de citas
- **Programa 299**. Sistema médico con estadísticas
- **Programa 300**. Sistema médico con login de usuarios

Enunciado

Enunciado de Ejercicios en VB.NET FORM

Programa 1. Formulario con un botón que muestra un mensaje Explicación: Este programa consiste en una ventana (formulario) que contiene un botón. Al hacer clic en este botón, se despliega una pequeña ventana de mensaje (MessageBox) que muestra "Hola, este es un mensaje".

Programa 2. Formulario con una etiqueta y un botón que cambia el texto Explicación: Este programa presenta un formulario con una etiqueta que inicialmente muestra "Texto inicial". Al pulsar un botón, el texto de la etiqueta se modifica a "Texto cambiado".

Programa 3. Formulario con un cuadro de texto y un botón para mostrar su contenido Explicación: Este programa incluye un formulario con un cuadro de texto donde el usuario puede introducir información. Al hacer clic en un botón, el contenido escrito en el cuadro de texto se muestra en una ventana de mensaje (MessageBox).

Programa 4. Formulario con dos botones que cambian el color de fondo Explicación: Este programa utiliza un formulario que tiene dos botones. Uno de los botones, al ser presionado, cambia el color de fondo del formulario a rojo, mientras que el otro botón lo cambia a azul.

Programa 5. Formulario con un cuadro de texto que convierte a mayúsculas Explicación: Este programa consiste en un formulario con un cuadro de texto y un botón. Cuando el usuario escribe texto en el cuadro y pulsa el botón, el texto introducido se convierte y se muestra en mayúsculas dentro del mismo cuadro de texto.

Programa 6. Formulario con dos cuadros de texto para sumar números Explicación: Este programa ofrece un formulario con dos cuadros de texto para que el usuario introduzca números, un botón para realizar la suma y una etiqueta para mostrar el resultado. Al hacer clic en el botón "Sumar", se obtienen los valores de ambos cuadros de texto, se suman y el resultado se exhibe en la etiqueta "lblResultado".

Programa 7. Formulario con un botón para limpiar cuadros de texto Explicación: Este programa contiene un formulario con dos cuadros de texto y un botón. Al pulsar el botón "Limpiar", el contenido de ambos cuadros de texto se borra, dejándolos vacíos.

Programa 8. Formulario con un botón que cierra la aplicación Explicación: Este programa consiste en un formulario con un único botón. Al hacer clic en este botón, la aplicación completa se cierra.

Programa 9. Formulario con una etiqueta que muestra la hora actual Explicación: Este programa tiene un formulario con una etiqueta y un temporizador. La etiqueta muestra continuamente la hora actual, la cual se actualiza cada segundo gracias al temporizador configurado.

Programa 10. Formulario con un cuadro de texto que cuenta caracteres Explicación: Este programa incluye un formulario con un cuadro de texto multilinea y una etiqueta. A medida que el usuario escribe en el cuadro de texto, la etiqueta actualiza dinámicamente el número de caracteres presentes en el texto.

Programa 11. Formulario con dos botones para mostrar imágenes diferentes Explicación: Este programa presenta un formulario con un PictureBox y dos botones. Cada botón está configurado para cargar y mostrar una imagen distinta ("imagen1.jpg" o "imagen2.jpg") en el PictureBox al ser pulsado.

Programa 12. Formulario con un botón que cambia el tamaño de la ventana Explicación: Este programa tiene un formulario con un botón. Al hacer clic en este botón, el tamaño del formulario (ventana) se ajusta a un ancho de 800 píxeles y un alto de 600 píxeles.

Programa 13. Formulario con un cuadro de texto que solo permite números Explicación: Este programa consiste en un formulario con un cuadro de texto diseñado para aceptar exclusivamente la entrada de caracteres numéricos. Cualquier otro carácter que el usuario intente introducir será ignorado.

Programa 14. Formulario con tres botones que cambian el color de texto Explicación: Este programa cuenta con un formulario que incluye una etiqueta de texto y tres botones. Cada botón permite cambiar el color del texto de la etiqueta a rojo, verde o azul, respectivamente, al ser pulsado.

Programa 15. Formulario con un cuadro de texto y un botón para invertir texto Explicación: Este programa presenta un formulario con un cuadro de texto y un botón. Al ingresar texto y presionar el botón "Invertir", el contenido del cuadro de texto se muestra en orden inverso.

Programa 16. Formulario con un contador de clics Explicación: Este programa dispone de un formulario con una etiqueta y un botón. Cada vez que se hace clic en el botón "Contar Clic", un contador interno se incrementa y la etiqueta "lblContador" muestra el número total de clics realizados.

Programa 17. Formulario con dos botones para mostrar mensajes distintos Explicación: Este programa presenta un formulario con dos botones. Al hacer clic en "Mensaje 1" se muestra un cuadro de diálogo con "Este es el primer mensaje", y al hacer clic en "Mensaje 2" se muestra "Este es el segundo mensaje".

Programa 18. Formulario con un CheckBox que habilita o deshabilita un botón Explicación: Este programa cuenta con un formulario que incluye un CheckBox y un botón inicialmente deshabilitado. La funcionalidad del botón "btnAccion" se activa o desactiva según el estado (marcado o desmarcado) del CheckBox.

Programa 19. Formulario con un TextBox multilinea y un botón para contar líneas Explicación: Este programa presenta un formulario con un cuadro de texto que acepta múltiples líneas de texto y un botón. Al hacer clic en el botón "Contar Líneas", se calcula y muestra en una etiqueta el número total de líneas de texto introducidas en el cuadro.

Programa 20. Formulario con una etiqueta que cambia de color al pasar el mouse Explicación: Este programa consiste en un formulario con una etiqueta de texto. Cuando el cursor del mouse se posiciona sobre la etiqueta, su color cambia a rojo, y cuando el cursor se retira, vuelve a su color original (negro).

Programa 21. Formulario con un botón que cambia el ícono del formulario Explicación: Este programa presenta un formulario con un botón. Al hacer clic en el botón "Cambiar Ícono", el ícono de la ventana del formulario se actualiza a una imagen especificada por "icono.ico".

Programa 22. Formulario con un TrackBar que controla el tamaño del texto Explicación: Este programa incluye un formulario con una etiqueta de texto y un control deslizante (TrackBar). Al mover el deslizador, el tamaño de la fuente del texto en la etiqueta cambia dinámicamente según el valor seleccionado en el TrackBar.

Programa 23. Formulario con un cuadro de texto que valida correo electrónico Explicación: Este programa presenta un formulario con un cuadro de texto para introducir un correo electrónico y un botón. Al hacer clic en "Validar", el programa utiliza expresiones regulares para verificar si el texto introducido tiene un formato de correo electrónico válido, mostrando el resultado en una etiqueta.

Programa 24. Formulario con un botón que rota imágenes Explicación: Este programa contiene un formulario con un PictureBox que muestra una imagen inicial y un botón. Al pulsar el botón "Rotar", la imagen mostrada en el PictureBox gira 90 grados.

Programa 25. Formulario con una lista de opciones en un ComboBox Explicación: Este programa consiste en un formulario con un ComboBox (lista desplegable). Al cargar el formulario, se añaden automáticamente tres opciones predefinidas ("Opción 1", "Opción 2", "Opción 3") a la lista desplegable.

Programa 26. Formulario con un ListBox que agrega elementos desde un TextBox Explicación: Este programa presenta un formulario con un cuadro de texto, un botón y un ListBox. El usuario puede escribir un elemento en el cuadro de texto y, al hacer clic en "Agregar", ese elemento se añade a la lista en el ListBox, y el cuadro de texto se limpia.

Programa 27. Formulario con un botón que elimina el elemento seleccionado del ListBox Explicación: Este programa cuenta con un formulario que contiene un ListBox y un botón. Si el usuario selecciona un elemento en el ListBox y pulsa el botón "Eliminar Seleccionado", el elemento elegido es retirado de la lista.

Programa 28. Formulario con un cuadro de texto que convierte a número romano Explicación: Este programa permite al usuario introducir un número entero en un cuadro de texto. Al hacer clic en "Convertir", el número se transforma a su representación en números romanos y el resultado se muestra en una etiqueta. Si la entrada no es un número válido, se indica un error.

Programa 29. Formulario con un botón que genera un número aleatorio Explicación: Este programa consiste en un formulario con un botón y una etiqueta. Al presionar el botón "Generar", se crea un número entero aleatorio entre 1 y 100, y este número se muestra en la etiqueta "lblNumero".

Programa 30. Formulario con un temporizador que actualiza la hora Explicación: Este programa dispone de un formulario con una etiqueta y un temporizador. La etiqueta "lblHora" muestra la hora actual, y el temporizador se encarga de actualizarla cada segundo, manteniendo la visualización en tiempo real.

Programa 31. Formulario con un botón que oculta y muestra una etiqueta Explicación: Este programa incluye un formulario con una etiqueta y un botón. Al hacer clic en "Ocultar/Mostrar", la visibilidad de la etiqueta se alterna, es decir, si está visible se oculta, y si está oculta se vuelve visible.

Programa 32. Formulario con un botón que abre un nuevo formulario vacío Explicación: Este programa presenta un formulario principal con un botón. Al pulsar el botón "Abrir Formulario", se crea y se muestra una nueva ventana de formulario completamente vacía, con el título "Nuevo Formulario".

Programa 33. Formulario con una barra de progreso que avanza al presionar un botón Explicación: Este programa contiene un formulario con una barra de progreso y un botón. Cada vez que se hace clic en el botón "Avanzar", el valor de la barra de progreso aumenta en 10 unidades, hasta alcanzar su valor máximo.

Programa 34. Formulario con un botón que copia texto al portapapeles Explicación: Este programa tiene un formulario con un cuadro de texto y un botón. Al introducir texto en el cuadro y presionar el botón "Copiar", el contenido del cuadro de texto se guarda en el portapapeles del sistema.

Programa 35. Formulario con un cuadro de texto que limita caracteres Explicación: Este programa consiste en un formulario con un cuadro de texto y una etiqueta de aviso. El cuadro de texto está configurado para aceptar un máximo de 10 caracteres, una limitación que se aplica directamente mediante una propiedad del control sin necesidad de código adicional.

Programa 36. Formulario con un botón que abre un cuadro de diálogo de color Explicación: Este programa presenta un formulario con un botón y una etiqueta. Al hacer clic en el botón "Elegir Color", se abre un cuadro de diálogo de selección de color, y el color elegido por el usuario se aplica al texto de la etiqueta.

Programa 37. Formulario con un botón que abre un cuadro de diálogo de fuente Explicación: Este programa incluye un formulario con un botón y una etiqueta. Al pulsar el botón "Elegir Fuente", se abre un cuadro de diálogo de selección de fuente, y la fuente y estilo seleccionados por el usuario se aplican al texto de la etiqueta.

Programa 38. Formulario con un botón que abre un cuadro de diálogo de archivo Explicación: Este programa contiene un formulario con un botón. Al hacer clic en "Abrir Archivo", se despliega un cuadro de diálogo para seleccionar un archivo. Una vez seleccionado, el nombre completo (ruta) del archivo se muestra en un cuadro de mensaje.

Programa 39. Formulario con un botón que abre un cuadro de diálogo de carpeta Explicación: Este programa consiste en un formulario con un botón. Al presionar "Elegir Carpeta", se abre un cuadro de diálogo que permite al usuario seleccionar una carpeta. La ruta de la carpeta seleccionada se muestra posteriormente en un cuadro de mensaje.

Programa 40. Formulario con un menú simple con dos opciones Explicación: Este programa presenta un formulario con una barra de menú que contiene dos opciones. Al hacer clic en "Opción 1" o "Opción 2" del menú, se muestra un mensaje correspondiente en un cuadro de diálogo.

Programa 41. Formulario con un menú que cambia el color de fondo Explicación: Este programa incluye un formulario con una barra de menú que ofrece opciones para cambiar el color de fondo del formulario. Las opciones "Rojo", "Verde" y "Azul" aplican el color respectivo al fondo de la ventana al ser seleccionadas.

Programa 42. Formulario con un menú que muestra un mensaje Explicación: Este programa consiste en un formulario con una barra de menú que incluye una opción llamada "Mostrar Mensaje". Al hacer clic en esta opción del menú, se despliega un cuadro de diálogo con el texto "Este es un mensaje desde el menú".

Programa 43. Formulario con un menú para abrir otro formulario Explicación: Este programa cuenta con un formulario principal que tiene un menú. Al seleccionar la opción "Abrir Formulario" del menú, se instancia y se muestra una nueva ventana de formulario (Form) con el título "Nuevo Formulario".

Programa 44. Formulario con una barra de herramientas (ToolStrip) Explicación: Este programa presenta un formulario con una barra de herramientas que contiene dos botones. Al hacer clic en "Nuevo", se muestra un mensaje informativo, y al hacer clic en "Salir", la aplicación se cierra.

Programa 45. Formulario con un cuadro de texto que busca en un ListBox Explicación: Este programa incluye un formulario con un cuadro de texto, un botón y un ListBox. El usuario puede introducir texto en "txtBuscar" y al hacer clic en "Buscar", el programa busca el elemento en el ListBox. Si lo encuentra, lo selecciona; si no, muestra un mensaje de "Elemento no encontrado".

Programa 46. Formulario con un botón que cambia el título del formulario Explicación: Este programa consiste en un formulario con un botón. Al presionar el botón "Cambiar Título", el texto de la barra de título del formulario se actualiza a "Nuevo Título del Formulario".

Programa 47. Formulario con dos botones que mueven un Label en el formulario Explicación: Este programa presenta un formulario con una etiqueta de texto y dos botones. Al hacer clic en "Izquierda", la etiqueta se mueve 10 píxeles hacia la izquierda, y al hacer clic en "Derecha", se mueve 10 píxeles hacia la derecha.

Programa 48. Formulario con un botón que muestra la fecha actual Explicación: Este programa contiene un formulario con un botón y una etiqueta. Al pulsar el botón "Mostrar Fecha", la etiqueta "lblFecha" actualiza su texto para mostrar la fecha actual en formato largo.

Programa 49. Formulario con un PictureBox que muestra diferentes imágenes Explicación: Este programa incluye un formulario con un PictureBox y dos botones. Cada botón permite cargar y visualizar una imagen distinta ("imagen1.jpg" o "imagen2.jpg") en el PictureBox al ser presionado, permitiendo al usuario alternar entre ellas.

Programa 50. Formulario con un botón que guarda texto en un archivo Explicación: Este programa presenta un formulario con un cuadro de texto multilinea y un botón. Al hacer clic en "Guardar", se abre un cuadro de diálogo para seleccionar la ubicación y el nombre del archivo, y el contenido del cuadro de texto se guarda en dicho archivo.

Programa 51. Agenda de contactos básica con ListBox Explicación: Este programa proporciona un formulario con un cuadro de texto para introducir nombres, un botón para añadir y un ListBox para mostrar los contactos. Al escribir un nombre y hacer clic en "Agregar", el nombre se añade a la lista y el cuadro de texto se limpia.

Programa 52. Calculadora simple (suma, resta, multiplicación, división) Explicación: Este programa simula una calculadora con dos cuadros de texto para números, y botones para las operaciones básicas (suma, resta, multiplicación, división). El resultado de la operación seleccionada se muestra en una etiqueta. Incluye validación básica para evitar división por cero.

Programa 53. Conversor de temperatura (Celsius-Fahrenheit) Explicación: Este programa permite al usuario introducir una temperatura en Celsius en un cuadro de texto. Al pulsar el botón "Convertir", calcula y muestra la equivalencia en grados Fahrenheit en una etiqueta.

Programa 54. Conversor de divisas Explicación: Este programa convierte una cantidad de Euros a Dólares Estadounidenses. El usuario introduce la cantidad en Euros en un cuadro de texto y, al hacer clic en "Convertir a USD", el resultado se muestra en una etiqueta utilizando una tasa de cambio predefinida (ej. 1.1).

Programa 55. Bloc de notas básico Explicación: Este programa es un formulario simple que contiene un cuadro de texto multilinea que ocupa todo el espacio disponible. Permite al usuario escribir y editar texto sin funcionalidad adicional de guardar o abrir, actuando como un bloc de notas minimalista.

Programa 56. Aplicación que guarda y carga texto desde archivo Explicación: Este programa es un editor de texto básico que permite al usuario escribir en un cuadro de texto. Dispone de botones para "Guardar" y "Cargar" contenido, utilizando cuadros de diálogo estándar para seleccionar archivos y guardar o recuperar el texto.

Programa 57. Aplicación que cuenta palabras en un TextBox Explicación: Este programa ofrece un formulario con un cuadro de texto multilinea y un botón. Al hacer clic en "Contar Palabras", el programa procesa el texto introducido y muestra en una etiqueta el número total de palabras que contiene.

Programa 58. Aplicación que muestra tablas de multiplicar Explicación: Este programa permite al usuario introducir un número en un cuadro de texto. Al hacer clic en "Generar Tabla", se calcula y se muestra en un ListBox la tabla de multiplicar de ese número del 1 al 10.

Programa 59. Aplicación que genera contraseñas aleatorias Explicación: Este programa cuenta con un botón que, al ser pulsado, genera una contraseña aleatoria de 8 caracteres. La contraseña está compuesta por letras mayúsculas, minúsculas, números y símbolos, y se muestra en una etiqueta.

Programa 60. Aplicación que muestra mensajes aleatorios Explicación: Este programa presenta un botón y una etiqueta. Al hacer clic en "Mostrar Mensaje", la etiqueta muestra uno de varios mensajes predefinidos de forma aleatoria, como "Hola!", "Bienvenido", etc..

Programa 61. Aplicación que calcula el IMC Explicación: Este programa permite calcular el Índice de Masa Corporal (IMC). El usuario introduce su peso en kg y su altura en metros en cuadros de texto, y al hacer clic en "Calcular IMC", el resultado se muestra en una etiqueta, con validación para altura cero.

Programa 62. Conversor de unidades de longitud (metros a cm, mm, km) Explicación: Este programa convierte una cantidad de metros introducida por el usuario a centímetros, milímetros y kilómetros. Los resultados de cada conversión se muestran en un ListBox.

Programa 63. Conversor de unidades de peso (kg a g, lb) Explicación: Este programa convierte una cantidad de kilogramos introducida por el usuario a gramos y libras. Los resultados de cada conversión se muestran en un ListBox.

Programa 64. Conversor de unidades de tiempo (horas a minutos y segundos) Explicación: Este programa permite al usuario introducir una cantidad de horas. Al hacer clic en "Convertir", calcula y muestra en una etiqueta la equivalencia de esas horas en minutos y segundos.

Programa 65. Conversor de unidades de velocidad (km/h a m/s) Explicación: Este programa convierte una velocidad dada en kilómetros por hora (km/h) a metros por segundo (m/s). El usuario introduce el valor en km/h, y el resultado de la conversión se muestra formateado en una etiqueta.

Programa 66. Aplicación que guarda configuraciones en My.Settings Explicación: Este programa demuestra cómo guardar y cargar datos de configuración utilizando `My.Settings`. El usuario introduce un dato en un cuadro de texto, lo guarda con un botón y luego puede cargarlo para que se muestre en una etiqueta. Requiere configuración previa de una propiedad "MiDato" en las propiedades del proyecto.

Programa 67. Calculadora de área de figuras geométricas (círculo y cuadrado) Explicación: Este programa calcula el área de un círculo y un cuadrado. El usuario introduce el radio y el lado en cuadros de texto, y al hacer clic en "Calcular Áreas", los resultados formateados se muestran en una etiqueta.

Programa 68. Calculadora de perímetro de figuras geométricas (círculo y cuadrado) Explicación: Similar al programa 67, este programa calcula el perímetro de un círculo y un cuadrado. El usuario introduce el radio y el lado, y los perímetros calculados se muestran en una etiqueta.

Programa 69. Juego del número secreto Explicación: Este es un juego de adivinanza donde el programa genera un número secreto entre 1 y 100 al cargar el formulario. El usuario introduce un intento en un cuadro de texto y el programa le indica si el número secreto es mayor, menor o si ha acertado.

Programa 70. Juego de adivinar palabra Explicación: Este programa es un juego de adivinanza de palabras. Se define una palabra secreta ("GATO"), y el usuario intenta adivinarla letra por letra. La etiqueta "lblPista" muestra la palabra con las letras adivinadas y guiones para las no adivinadas. Si el usuario adivina la palabra, se muestra un mensaje de "¡Ganaste!".

Programa 71. Juego de piedra, papel o tijera Explicación: Este programa simula el juego de piedra, papel o tijera contra el ordenador. El usuario selecciona una opción de un ComboBox, y el programa elige una aleatoriamente para el PC, mostrando el resultado (Ganaste, Perdiste, Empate) y la elección del PC en una etiqueta.

Programa 72. Cronómetro con Start, Stop y Reset Explicación: Este programa implementa un cronómetro con controles para "Start", "Stop" y "Reset". Un temporizador interno incrementa los segundos, y la etiqueta "lblTiempo" muestra el tiempo transcurrido en formato hh:mm:ss. Los botones permiten controlar el inicio, la pausa y el reinicio del cronómetro.

Programa 73. Temporizador con alarma Explicación: Este programa funciona como un temporizador. El usuario introduce una cantidad de segundos, y al iniciar, una etiqueta muestra la cuenta regresiva. Cuando el tiempo llega a cero, el temporizador se detiene y se muestra un mensaje de "¡Tiempo terminado!".

Programa 74. Editor de listas (agregar, modificar y borrar) Explicación: Este programa permite gestionar una lista de elementos en un ListBox. Incluye funcionalidades para "Agregar" un nuevo elemento desde un cuadro de texto, "Modificar" el elemento seleccionado en la lista, y "Borrar" el elemento seleccionado.

Programa 75. Editor de texto con guardar y abrir archivo Explicación: Este programa es un editor de texto básico integrado en un formulario. Utiliza un menú con opciones "Archivo → Abrir" y "Archivo → Guardar" para interactuar con el sistema de archivos, permitiendo al usuario cargar contenido de un archivo a un TextBox o guardar el texto actual en un archivo.

Programa 76. Aplicación con varios formularios enlazados Explicación: Este programa demuestra la interacción entre múltiples formularios. El formulario principal (Form1) tiene un botón que, al ser pulsado, abre y muestra una segunda ventana de formulario (Form2).

Programa 77. Agenda que guarda datos en archivo de texto Explicación: Este programa simula una agenda que guarda contactos (nombre y teléfono) en un archivo de texto llamado "agenda.txt". Los contactos agregados también se muestran en un ListBox en el formulario.

Programa 78. Agenda que guarda datos en archivo CSV Explicación: Similar al programa anterior, esta agenda guarda contactos (nombre y teléfono) en un archivo CSV (valores separados por coma) llamado "agenda.csv". Los contactos también se visualizan en un ListBox.

Programa 79. Agenda que guarda datos en archivo XML Explicación: Esta agenda permite guardar contactos (nombre y teléfono) en un archivo XML llamado "agenda.xml". El programa maneja la creación del archivo XML si no existe y añade nuevos contactos como atributos de elementos "Contacto", mostrando también los contactos en un ListBox.

Programa 80. Agenda que guarda datos en base de datos Access Explicación: Esta agenda guarda contactos (nombre y teléfono) en una base de datos Microsoft Access ("agenda.accdb"). Utiliza `OleDbConnection` para conectar y `INSERT` para agregar datos a una tabla "Contactos". Los contactos también se añaden a un ListBox. Requiere una base de datos Access preexistente.

Programa 81. Formulario con gráficos simples Explicación: Este programa demuestra cómo dibujar gráficos básicos en un PictureBox. Al hacer clic en "Dibujar", se trazan una línea azul, un rectángulo rojo y una elipse verde sobre un fondo blanco en el PictureBox.

Programa 82. Formulario con impresión de texto Explicación: Este programa permite al usuario introducir texto en un cuadro de texto y luego imprimirlo. Al hacer clic en "Imprimir", se abre un cuadro de diálogo de impresión, y si el usuario confirma, el texto se envía a la impresora utilizando un `PrintDocument`.

Programa 83. Calculadora de ecuaciones de segundo grado Explicación: Este programa calcula las soluciones (raíces) de una ecuación de segundo grado ($Ax^2 + Bx + C = 0$). El usuario introduce los coeficientes A, B y C en cuadros de texto, y al hacer clic en "Calcular", se muestran las soluciones x1 y x2 en una etiqueta, o un mensaje si no hay soluciones reales.

Programa 84. Calculadora de interés simple Explicación: Este programa calcula el interés simple. El usuario introduce el capital inicial, la tasa de interés (como porcentaje) y el tiempo en años. Al hacer clic en "Calcular", el interés simple se muestra en una etiqueta.

Programa 85. Calculadora de interés compuesto Explicación: Este programa calcula el monto final con interés compuesto. El usuario introduce el capital inicial, la tasa de interés (como porcentaje) y el tiempo en años. Al hacer clic en "Calcular", el monto total con interés compuesto se muestra formateado en una etiqueta.

Programa 86. Conversor de números decimales a binarios Explicación: Este programa convierte un número decimal entero a su representación binaria. El usuario introduce el número decimal en un cuadro de texto y, al hacer clic en "Convertir", el resultado binario se muestra en una etiqueta.

Programa 87. Conversor de binarios a decimales Explicación: Este programa convierte un número binario a su equivalente decimal. El usuario introduce el número binario en un cuadro de texto y, al hacer clic en "Convertir", el resultado decimal se muestra en una etiqueta. Incluye manejo de errores para entradas no válidas.

Programa 88. Conversor de números romanos a enteros Explicación: Este programa convierte un número romano a su valor entero decimal. El usuario introduce el número romano en un cuadro de texto y, al hacer clic en "Convertir", el valor entero se muestra en una etiqueta. Utiliza un diccionario para mapear los símbolos romanos a sus valores.

Programa 89. Conversor de horas a minutos y segundos Explicación: Este programa convierte una cantidad de horas a su equivalencia en minutos y segundos. El usuario introduce el número de horas en un cuadro de texto, y el resultado se muestra en una etiqueta.

Programa 90. Conversor de kilogramos a libras Explicación: Este programa convierte una cantidad de kilogramos a libras. El usuario introduce el valor en kg, y al hacer clic en "Convertir", el resultado en libras se muestra formateado en una etiqueta.

Programa 91. Conversor de metros a pies Explicación: Este programa convierte una cantidad de metros a pies. El usuario introduce el valor en metros, y al hacer clic en "Convertir", el resultado en pies se muestra formateado en una etiqueta.

Programa 92. Conversor de Celsius a Kelvin Explicación: Este programa convierte una temperatura de grados Celsius a Kelvin. El usuario introduce la temperatura en Celsius, y el resultado en Kelvin se muestra formateado en una etiqueta.

Programa 93. Conversor de binario a hexadecimal Explicación: Este programa convierte un número binario a su representación hexadecimal. El usuario introduce el número binario y, al hacer clic en "Convertir", el resultado hexadecimal se muestra en una etiqueta. Incluye manejo de errores para entradas binarias inválidas.

Programa 94. Conversor de hexadecimal a decimal Explicación: Este programa convierte un número hexadecimal a su equivalente decimal. El usuario introduce el valor hexadecimal y, al hacer clic en "Convertir", el resultado decimal se muestra en una etiqueta. También maneja errores para entradas hexadecimales inválidas.

Programa 95. Juego de memoria con botones Explicación: Este programa presenta un juego de memoria con cuatro botones que tienen colores predefinidos (rojo, verde, azul, amarillo). Los botones deben ser presionados en un orden específico (secuencia) que el juego define. Si el usuario presiona el botón correcto en la secuencia, el juego avanza; si no, se reinicia con un mensaje de error.

Programa 96. Juego de laberinto simple Explicación: Este programa simula un juego de laberinto simple. El usuario controla un "jugador" (representado por un botón) usando las teclas de flecha. El objetivo es mover al jugador hasta una "meta" (otro botón). Si el jugador intersecta con la meta, se muestra un mensaje de "¡Has ganado!". (Requiere que la propiedad `KeyPreview` del formulario esté en `True`).

Programa 97. Juego de dados con imágenes Explicación: Este programa simula el lanzamiento de dos dados. Al hacer clic en "Tirar Dados", se generan dos números aleatorios entre 1 y 6, y se muestran las imágenes correspondientes a esos números en dos PictureBox. (Requiere que las imágenes "dado1.jpg" a "dado6.jpg" estén en los recursos del proyecto).

Programa 98. Juego de ahorcado Explicación: Este programa implementa un juego de ahorcado. El usuario intenta adivinar una palabra secreta ("VBNET") letra por letra. La etiqueta "lblPalabra" muestra las letras adivinadas y guiones para las no adivinadas. Los intentos fallidos se cuentan (6 intentos). El juego informa si el usuario gana o pierde.

Programa 99. Juego de tic-tac-toe Explicación: Este programa simula un juego de tres en raya (tic-tac-toe) con nueve botones que representan el tablero. Los jugadores se turnan para marcar "X" u "O". El programa gestiona los turnos y permite a los jugadores hacer movimientos en las celdas vacías.

Programa 100. Juego de cartas simple Explicación: Este programa simula un juego de cartas básico. Al hacer clic en "Sacar Carta", se selecciona una carta aleatoria de un mazo predefinido y su imagen se muestra en un PictureBox. (Requiere que las imágenes "c1.jpg" a "c5.jpg" estén en los recursos del proyecto).

Programa 101. Juego de lotería con números aleatorios Explicación: Este programa simula un sorteo de lotería. Al hacer clic en "Generar Números", se generan 6 números aleatorios únicos entre 1 y 49, los cuales se muestran en un ListBox.

Programa 102. Juego de ruleta simple Explicación: Este programa simula una ruleta simple. Al hacer clic en "Girar", se genera un número aleatorio entre 0 y 36, representando el resultado de la ruleta, y se muestra en una etiqueta.

Programa 103. Juego de preguntas y respuestas Explicación: Este programa presenta un juego de preguntas donde se muestra una pregunta predefinida ("Capital de Francia?"). El usuario introduce una respuesta en un cuadro de texto y al verificar, el programa indica si la respuesta es correcta o incorrecta.

Programa 104. Juego de carreras con imágenes Explicación: Este programa simula una carrera entre dos "carros" representados por PictureBox. Al hacer clic en "Correr", un temporizador mueve los carros aleatoriamente. La carrera termina cuando uno de los carros llega al final del formulario.

Programa 105. Juego de disparos con botones Explicación: Este programa simula un juego de disparos simple. Al hacer clic en "Disparar", el programa pregunta si se "acertó" (con un MessageBox Yes/No). Si el usuario selecciona "Sí", el puntaje se incrementa y se muestra en una etiqueta.

Programa 106. Juego de laberinto con teclas de dirección Explicación: Similar al programa 96, este juego permite mover un "jugador" (botón) con las teclas de dirección (arriba, abajo, izquierda, derecha) a través de un panel que simula un laberinto hasta llegar a una "meta" (otro botón). Al alcanzarla, se notifica la victoria. (Requiere `KeyPreview=True` en el Form).

Programa 107. Agenda con búsqueda por nombre Explicación: Este programa implementa una agenda que permite buscar contactos por nombre. Contiene una lista predefinida de contactos. El usuario introduce un filtro en un cuadro de texto, y al buscar, el ListBox muestra solo los contactos que contienen el texto del filtro.

Programa 108. Agenda con edición de contactos Explicación: Este programa permite editar contactos en un ListBox. El usuario selecciona un contacto en la lista, modifica el texto en un cuadro de texto y, al hacer clic en "Editar", el elemento seleccionado se actualiza con el nuevo texto.

Programa 109. Agenda con borrado de contactos Explicación: Este programa facilita la eliminación de contactos de un ListBox. Si el usuario selecciona un elemento en la lista y pulsa el botón "Borrar", el contacto seleccionado es removido.

Programa 110. Agenda con exportación a archivo de texto Explicación: Este programa permite exportar la lista de contactos de un ListBox a un archivo de texto llamado "agenda.txt". Al hacer clic en "Exportar", todos los elementos de la lista se escriben en el archivo, cada uno en una línea nueva.

Programa 111. Agenda con importación desde archivo de texto Explicación: Este programa permite importar contactos desde un archivo de texto a un ListBox. Al hacer clic en "Importar", se abre un cuadro de diálogo para seleccionar un archivo. Una vez seleccionado, el contenido del archivo se carga y reemplaza los elementos actuales en el ListBox.

Programa 112. Agenda con exportación a Excel Explicación: Este programa permite exportar los elementos de un ListBox a una hoja de cálculo de Microsoft Excel. Al hacer clic en "Exportar a Excel", los contactos se escriben en la primera columna de un nuevo libro de Excel, y la aplicación de Excel se hace visible. (Requiere referencia a la biblioteca de objetos de Microsoft Excel).

Programa 113. Agenda con importación desde Excel Explicación: Este programa importa datos desde un archivo de Excel a un ListBox. El usuario selecciona un archivo de Excel, y el programa lee los valores de la primera columna de la primera hoja, agregándolos al ListBox. Luego, cierra el libro y la aplicación de Excel.

Programa 114. Agenda con base de datos SQLite Explicación: Este programa gestiona contactos utilizando una base de datos SQLite ("agenda.db"). Al cargar, crea la tabla "contactos" si no existe y carga los contactos existentes en un ListBox. Permite "Agregar" nuevos contactos desde un TextBox a la base de datos y actualizar la lista.

Programa 115. Agenda con base de datos SQL Server Explicación: Esta agenda interactúa con una base de datos SQL Server. Al cargar, establece conexión y carga contactos en un ListBox. Un botón "Agregar" permite insertar nuevos contactos desde un TextBox en la tabla "contactos" de la base de datos y luego refrescar la lista.

Programa 116. Agenda con base de datos MySQL Explicación: Este programa implementa una agenda que se conecta a una base de datos MySQL. Al iniciar, abre la conexión y carga los nombres de contactos existentes en un ListBox. Permite añadir nuevos contactos a la tabla "contactos" de MySQL a través de un botón y un TextBox.

Programa 117. Agenda con base de datos PostgreSQL Explicación: Esta agenda se conecta a una base de datos PostgreSQL. Al cargar, abre la conexión y recupera los nombres de contactos de la tabla "contactos" para mostrarlos en un ListBox. También permite "Agregar" nuevos contactos desde un TextBox a la base de datos.

Programa 118. Aplicación de notas con recordatorio Explicación: Este programa permite al usuario escribir una nota y asociarle una fecha y hora de recordatorio usando un `DateTimePicker`. Al guardar, la nota y su recordatorio se almacenan. Un temporizador verifica periódicamente si alguna alarma ha llegado, mostrando un `MessageBox` si es el caso.

Programa 119. Aplicación de calendario simple Explicación: Este programa presenta un calendario visual (`MonthCalendar`). Al seleccionar una fecha en el calendario, la etiqueta "lblFecha" actualiza su texto para mostrar la fecha elegida.

Programa 120. Agenda con calendario integrado Explicación: Este programa combina una agenda con un `MonthCalendar`. Permite al usuario seleccionar una fecha en el calendario y añadir una nota para esa fecha específica. Las notas se almacenan en un diccionario y se muestran en un ListBox, actualizándose al cambiar la fecha seleccionada.

Programa 121. Agenda con alarma Explicación: Este programa permite al usuario crear alarmas asociadas a una nota y una fecha/hora. Las alarmas se listan en un ListBox. Un temporizador verifica constantemente si alguna alarma ha llegado, mostrando un `MessageBox` y eliminando la alarma de la lista al activarse.

Programa 122. Agenda con notificaciones Explicación: Este programa permite añadir notas a un ListBox. Un temporizador, configurado para activarse cada minuto, muestra un `MessageBox` con una notificación indicando el número total de notas pendientes, actuando como un recordatorio general.

Programa 123. Agenda con múltiples formularios Explicación: Este programa demuestra la funcionalidad de abrir un formulario secundario desde el principal. El formulario principal tiene un botón que, al ser pulsado, instancia y muestra el "Formulario 2" (`Form2`).

Programa 124. Agenda con login de usuario Explicación: Este programa implementa un sistema de inicio de sesión básico. El usuario introduce un nombre de usuario y una contraseña (ocultada con `PasswordChar`). Si las credenciales ("admin"/"1234") son correctas, se muestra "Acceso concedido"; de lo contrario, "Usuario o contraseña incorrecta".

Programa 125. Agenda con niveles de usuario Explicación: Este programa simula un sistema de acceso con diferentes niveles de usuario (Usuario, Administrador) seleccionables mediante un `ComboBox`. Al hacer clic en "Login", se muestra un mensaje indicando el tipo de acceso concedido, sin una validación de contraseña real.

Programa 126. Agenda con encriptación de datos Explicación: Este programa demuestra una encriptación simple de texto utilizando Base64. El usuario introduce un dato en un cuadro de texto, y al hacer clic en "Encriptar", el texto se convierte a su representación Base64 y se muestra en una etiqueta.

Programa 127. Agenda con desencriptación de datos Explicación: Complementario al programa 126, este programa desencripta un texto codificado en Base64. El usuario introduce el texto encriptado en un cuadro de texto y, al hacer clic en "Desencriptar", el texto original se recupera y se muestra en una etiqueta.

Programa 128. Calculadora científica simple Explicación: Este programa es una calculadora básica con capacidad de realizar sumas, restas, multiplicaciones y divisiones. Dispone de una pantalla (TextBox), botones numéricos y botones de operación. Permite introducir dos valores y una operación, mostrando el resultado al presionar "=".

Programa 129. Calculadora con funciones trigonométricas Explicación: Esta calculadora extiende la funcionalidad para incluir operaciones trigonométricas básicas. El usuario introduce un número en la pantalla (TextBox) y, al presionar botones como "Sin", "Cos" o "Tan", se calcula y muestra el seno, coseno o tangente del valor, respectivamente.

Programa 130. Calculadora con funciones logarítmicas Explicación: Esta calculadora ofrece funciones logarítmicas. El usuario introduce un número en la pantalla y, al hacer clic en "Log" o "Ln", se calcula y muestra el logaritmo en base 10 o el logaritmo natural del valor introducido, respectivamente.

Programa 131. Calculadora con potencias y raíces Explicación: Esta calculadora permite realizar operaciones de potencia y raíz cuadrada. El usuario puede calcular la raíz cuadrada de un número directamente con "√" o realizar una potencia (`valor1 ^ valor2`) ingresando dos números y presionando el botón "igual" después de la operación "^".

Programa 132. Calculadora con historial de operaciones Explicación: Esta calculadora básica, además de realizar operaciones aritméticas, mantiene un historial de las operaciones realizadas. Cada vez que se completa un cálculo al presionar "=", la operación y su resultado se añaden a un ListBox para su registro.

Programa 133. Calculadora con memoria de resultados Explicación: Esta calculadora incorpora funciones de memoria (Memory Store, Memory Recall, Memory Clear). Permite almacenar el valor actual de la pantalla en una variable de memoria (MS), recuperar ese valor a la pantalla (MR) y borrar la memoria (MC).

Programa 134. Juego de preguntas con puntuación Explicación: Este programa es un juego de preguntas donde se presentan preguntas predefinidas secuencialmente. El usuario introduce una respuesta. Si es correcta, se incrementa un puntaje, que se muestra en una etiqueta. Al finalizar todas las preguntas, se muestra el puntaje total.

Programa 135. Juego de preguntas con tiempo límite Explicación: Este es un juego de preguntas con una pregunta fija. El usuario tiene un tiempo límite (10 segundos) para responder, que se muestra en una etiqueta y disminuye con un temporizador. Si el tiempo se agota antes de responder, se muestra un mensaje.

Programa 136. Juego de preguntas con niveles Explicación: Este programa presenta un juego de preguntas con diferentes niveles de dificultad seleccionables a través de un ComboBox (Fácil, Medio, Difícil). Al cambiar el nivel, se muestra una pregunta correspondiente a esa dificultad.

Programa 137. Juego de preguntas con base de datos Explicación: Este juego de preguntas almacena sus preguntas y respuestas en una base de datos SQLite. Al cargar, crea la tabla de preguntas si no existe y recupera la primera pregunta. El usuario introduce la respuesta y el programa verifica si es correcta.

Programa 138. Juego de preguntas con imágenes Explicación: En este juego, las preguntas se presentan visualmente a través de imágenes en un PictureBox. El usuario debe identificar lo que representa la imagen y escribir su respuesta en un TextBox. El programa verifica si la respuesta es correcta. (Requiere imágenes como "imagen1.jpg").

Programa 139. Juego de preguntas con sonidos Explicación: Este juego de preguntas incorpora elementos de audio. Al hacer clic en un botón, se reproduce un sonido. El usuario debe identificar el sonido y escribir su respuesta. El programa verifica la respuesta para determinar si es correcta. (Requiere un archivo de sonido "sonido1.wav").

Programa 140. Juego de preguntas multijugador Explicación: Este programa es un juego de preguntas diseñado para dos jugadores. Ambos jugadores introducen sus respuestas a una misma pregunta. El programa lleva la cuenta de los puntajes de cada jugador y los muestra en una etiqueta.

Programa 141. Juego de preguntas con ranking Explicación: Este juego de preguntas lleva un registro de la puntuación actual del jugador y la añade a una lista de puntajes. Los puntajes se muestran en un ListBox, creando un ranking simple. El juego avanza a la siguiente pregunta si la respuesta es correcta.

Programa 142. Juego de preguntas con exportación de resultados Explicación: Este programa permite exportar los resultados de un juego de preguntas (asumiendo que los resultados están en un ListBox) a un archivo de texto llamado "resultados.txt". Todos los elementos del ListBox se guardan como líneas en el archivo.

Programa 143. Juego de preguntas con importación de resultados Explicación: Este programa permite importar resultados de un juego de preguntas desde un archivo de texto. El usuario selecciona un archivo y su contenido se carga en un ListBox, reemplazando cualquier resultado previo.

Programa 144. Juego de preguntas con estadísticas Explicación: Este juego de preguntas rastrea y muestra estadísticas sobre el rendimiento del jugador, incluyendo el número de respuestas correctas y el total de intentos. Estas estadísticas se actualizan y se muestran en una etiqueta a medida que el juego avanza.

Programa 145. Juego de preguntas con gráficos Explicación: Este programa visualiza los resultados de un juego de preguntas mediante un gráfico de columnas. Al hacer clic en "Actualizar Gráfico", se genera un gráfico que muestra el número de respuestas "Correctas" e "Incorrectas". (Requiere el control `Chart` de `System.Windows.Forms.DataVisualization.Charting`).

Programa 146. Juego de preguntas con personalización Explicación: Este programa permite al usuario agregar sus propias preguntas a una lista. Al introducir una pregunta en un TextBox y hacer clic en "Agregar Pregunta", esta se añade a un ListBox de preguntas, permitiendo la personalización del juego.

Programa 147. Juego de preguntas con guardado automático Explicación: Este programa implementa un mecanismo de guardado automático para las respuestas del usuario. Un temporizador guarda el contenido del cuadro de texto de respuesta en un archivo ("respuestas.txt") cada 5 segundos. También se puede guardar manualmente con un botón.

Programa 148. Juego de preguntas con recuperación de partida Explicación: Este programa permite al usuario guardar y cargar el progreso de una partida de preguntas. Al guardar, se almacena el índice de la pregunta actual en un archivo ("partida.txt"). Al cargar, se recupera este índice para continuar el juego desde donde se dejó.

Programa 149. Juego de preguntas con dificultad progresiva Explicación: Este juego presenta preguntas con una dificultad que aumenta progresivamente. Inicialmente, muestra una pregunta fácil. Al verificar la respuesta, el juego avanza al siguiente nivel de dificultad (medio, difícil) mostrando una pregunta correspondiente, hasta que el juego termina.

Programa 150. Juego de preguntas con logros desbloqueables Explicación: Este juego de preguntas incluye un sistema de logros. A medida que el usuario responde correctamente, se pueden desbloquear "logros" (por ejemplo, "Primer acierto!", "Dos aciertos seguidos!"), los cuales se muestran en un ListBox.

Programa 151. Editor de imágenes simple Explicación: Este programa es un editor de imágenes básico que permite cargar una imagen en un PictureBox y aplicarle un filtro. Al hacer clic en "Convertir a Gris", la imagen cargada se transforma a escala de grises.

Programa 152. Editor de audio simple Explicación: Este programa permite cargar un archivo de audio (WAV) y reproducirlo. Al hacer clic en "Cargar Audio", el usuario selecciona un archivo, y luego puede reproducirlo con el botón "Reproducir".

Programa 153. Reproductor de música básico Explicación: Este programa es un reproductor de música simple que utiliza el control `AxWindowsMediaPlayer`. Permite al usuario "Cargar Música" desde un archivo, y luego "Play" para reproducirla y "Stop" para detenerla.

Programa 154. Reproductor de video básico Explicación: Este programa reproduce videos utilizando el control `AxWindowsMediaPlayer`. Al hacer clic en "Cargar Video", el usuario selecciona un archivo de video, y el programa lo reproduce automáticamente en el reproductor integrado.

Programa 155. Reproductor multimedia integrado Explicación: Este programa es un reproductor multimedia versátil que permite cargar y reproducir tanto archivos de audio (WAV, MP3) como de video (MP4, AVI) utilizando el control `AxWindowsMediaPlayer`, diferenciando los filtros de archivo para cada tipo de contenido.

Programa 156. Aplicación que controla volumen del sistema Explicación: Este programa utiliza un `TrackBar` para simular el control del volumen y muestra el valor en una etiqueta. Se menciona que el control real del volumen del sistema operativo requeriría el uso de APIs avanzadas o bibliotecas externas.

Programa 157. Aplicación que controla brillo de pantalla Explicación: Similar al control de volumen, este programa usa un `TrackBar` para simular el control de brillo de pantalla y muestra su valor. Se advierte que el ajuste real del brillo de la pantalla necesitaría WMI o APIs específicas del sistema operativo.

Programa 158. Aplicación que toma capturas de pantalla Explicación: Este programa permite al usuario tomar una captura de toda la pantalla principal. Al hacer clic en "Capturar Pantalla", el contenido visible de la pantalla se captura como una imagen y se muestra en un `PictureBox`.

Programa 159. Aplicación que graba sonido Explicación: Este programa permite grabar audio desde el micrófono. Al hacer clic en "Grabar", se inicia la grabación y se le pide al usuario que guarde el archivo WAV. Al hacer clic en "Detener", la grabación finaliza y el archivo se cierra. (Requiere la biblioteca NAudio).

Programa 160. Aplicación que graba video Explicación: Este programa conceptualiza la grabación de video desde una cámara web. Al hacer clic en "Grabar Video", se intenta iniciar la captura de video y mostrar los fotogramas en un `PictureBox`. El botón "Detener" detiene la captura. (Requiere librerías como AForge o EmguCV para una implementación real).

Programa 161. Aplicación que reproduce listas de reproducción Explicación: Este programa permite crear y reproducir listas de reproducción de música. El usuario puede "Agregar Canción" (archivos MP3/WAV) a un `ListBox`. Luego, al seleccionar una canción de la lista y hacer clic en "Reproducir", se reproduce usando `AxWindowsMediaPlayer`.

Programa 162. Aplicación que descarga archivos de internet Explicación: Este programa permite al usuario descargar archivos desde una URL de internet. El usuario introduce la URL y, al hacer clic en "Descargar", se abre un `SaveFileDialog` para elegir la ubicación y el nombre del archivo, tras lo cual el archivo se descarga.

Programa 163. Aplicación que sube archivos a servidor FTP Explicación: Este programa permite subir un archivo local a un servidor FTP. El usuario introduce la URL del servidor, usuario y contraseña, selecciona un archivo, y al hacer clic en "Subir Archivo", este se envía al servidor FTP.

Programa 164. Aplicación que se conecta a una API REST Explicación: Este programa demuestra cómo realizar una consulta a una API REST. El usuario introduce una URL, y al hacer clic en "Consultar", la aplicación realiza una solicitud HTTP GET y muestra la respuesta obtenida en un cuadro de texto multilinea.

Programa 165. Aplicación que se conecta a una API SOAP Explicación: Este programa conceptualiza la conexión a una API SOAP. El usuario introduciría la URL del servicio y al hacer clic en "Consultar", se invocaría un método del servicio web y el resultado se mostraría en un cuadro de texto. (Requiere una referencia de servicio SOAP agregada).

Programa 166. Aplicación que consume servicios web JSON Explicación: Este programa consume datos de servicios web que devuelven JSON. El usuario introduce una URL, y al hacer clic en "Consultar", se descarga la cadena JSON, se parsea usando `Newtonsoft.Json.Linq` y el objeto JSON resultante se muestra en un cuadro de texto.

Programa 167. Aplicación que consume servicios web XML Explicación: Este programa consume datos de servicios web que devuelven XML. El usuario introduce una URL y, al hacer clic en "Consultar", la aplicación descarga la cadena XML, la carga en un `XmlDocument` y muestra el contenido XML completo en un cuadro de texto.

Programa 168. Cliente de chat local simple Explicación: Este programa simula un cliente de chat local muy básico. El usuario escribe un mensaje en un `TextBox` y, al hacer clic en "Enviar", el mensaje se añade a un `ListBox` precedido por "Tú:", y el `TextBox` se limpia.

Programa 169. Cliente de chat en red local Explicación: Este programa conceptualiza un cliente de chat que puede enviar mensajes a través de una red local utilizando `TcpClient`. El usuario escribe un mensaje y al "Enviar", si hay una conexión establecida, el mensaje se envía a través del flujo de red y se muestra en el `ListBox` del chat.

Programa 170. Cliente de chat con base de datos Explicación: Este programa simula un cliente de chat que almacena mensajes en una base de datos SQLite ("chat.db"). Al cargar, crea la tabla de mensajes si no existe. Al "Enviar" un mensaje, este se inserta en la base de datos y se muestra en el `ListBox`.

Programa 171. Cliente de chat con login de usuario Explicación: Este programa incorpora un sistema de inicio de sesión para el cliente de chat. El usuario debe introducir un nombre de usuario y una contraseña. Si ambos campos están rellenos, se simula un login exitoso con un mensaje de bienvenida; de lo contrario, se solicita ingresar los datos.

Programa 172. Cliente de chat con historial de mensajes Explicación: Este programa permite al cliente de chat guardar y cargar el historial de mensajes. Al iniciar, carga los mensajes previos de un archivo ("historial.txt") en un `ListBox`. Cada nuevo mensaje enviado también se añade al archivo de historial.

Programa 173. Cliente de chat con emoticonos Explicación: Este programa mejora la experiencia del chat permitiendo al usuario adjuntar emoticonos. El usuario escribe un mensaje, selecciona un emoticono de un `ComboBox` y al enviar, el mensaje junto con el emoticono seleccionado se muestra en el `ListBox` del chat.

Programa 174. Cliente de chat con archivos adjuntos Explicación: Este programa permite al usuario adjuntar un archivo a su mensaje de chat. Al hacer clic en "Adjuntar Archivo", se abre un `OpenFileDialog` para seleccionar el archivo. Cuando se envía el mensaje, el texto del mensaje y la ruta del archivo adjunto se muestran en el `ListBox`.

Programa 175. Cliente de chat con múltiples salas Explicación: Este programa implementa un cliente de chat con soporte para múltiples salas (General, Juegos, Música) seleccionables a través de un `ComboBox`. Cada sala tiene su propio historial de mensajes, y al cambiar de sala, el `ListBox` muestra los mensajes correspondientes a esa sala.

Programa 176. Cliente de chat con videollamadas Explicación: Este programa conceptualiza la funcionalidad de videollamadas dentro de un cliente de chat. Al hacer clic en "Iniciar Videollamada", se muestra un mensaje indicando que la implementación real requeriría librerías especializadas como AForge o WebRTC.

Programa 177. Cliente de chat con llamadas de voz Explicación: Similar al programa de videollamadas, este programa conceptualiza las llamadas de voz. Los botones "Iniciar Llamada" y "Detener" simulan el control, pero la implementación real requeriría librerías como NAudio o soporte para VoIP.

Programa 178. Cliente de chat con notificaciones Explicación: Este programa utiliza un `NotifyIcon` para mostrar notificaciones al usuario. Al hacer clic en "Enviar Notificación", se muestra una burbuja de notificación en la bandeja del sistema con un mensaje de "Tienes un nuevo mensaje".

Programa 179. Cliente de chat con moderadores Explicación: Este programa conceptualiza la función de moderación. Permite bloquear usuarios introduciendo su nombre en un `TextBox` y haciendo clic en "Bloquear Usuario", añadiéndolo a una lista de usuarios bloqueados.

Programa 180. Cliente de chat con administración Explicación: Este programa simula funciones de administración de usuarios para un chat. Permite "Agregar Usuario" a una lista (`ListBox`) desde un `TextBox` y "Eliminar Usuario" seleccionando uno de la lista.

Programa 181. Cliente de chat con cifrado de mensajes Explicación: Este programa implementa un cifrado básico de mensajes usando Base64. El usuario escribe un mensaje, y al "Enviar", el texto se cifra a Base64 y se muestra en el `ListBox` del chat en su formato cifrado.

Programa 182. Cliente de chat con recuperación de historial Explicación: Este programa guarda los mensajes del chat en un archivo ("historial_chat.txt"). Además de enviar mensajes y guardarlos, cuenta con un botón "Recuperar Historial" que carga todos los mensajes guardados en el `ListBox`.

Programa 183. Cliente de chat con estadísticas Explicación: Este programa proporciona una función para mostrar estadísticas básicas del chat. Al hacer clic en "Mostrar Estadísticas", se muestra un `MessageBox` con el número total de mensajes registrados en el `ListBox`.

Programa 184. Cliente de chat con exportación de mensajes Explicación: Este programa permite exportar el historial de mensajes del chat. Al hacer clic en "Exportar", se abre un `SaveFileDialog` para que el usuario elija la ubicación y el nombre del archivo, y luego todos los mensajes del `ListBox` se guardan en el archivo seleccionado.

Programa 185. Cliente de chat con importación de mensajes Explicación: Este programa permite importar mensajes de chat desde un archivo. El usuario selecciona un archivo con un `OpenFileDialog`, y su contenido se carga en el `ListBox` del chat, reemplazando los mensajes actuales.

Programa 186. Cliente de correo electrónico básico Explicación: Este programa es un cliente de correo electrónico básico que permite al usuario componer un mensaje (para, asunto, cuerpo). Al hacer clic en "Enviar", se crea un objeto `MailMessage` con los datos, pero se muestra un mensaje indicando que el envío real requeriría una configuración SMTP.

Programa 187. Cliente de correo con envío SMTP Explicación: Este programa permite enviar correos electrónicos a través de un servidor SMTP. El usuario introduce el destinatario, asunto, cuerpo, servidor SMTP, su usuario y contraseña. Al hacer clic en "Enviar", el programa se conecta al servidor SMTP y envía el correo.

Programa 188. Cliente de correo con recepción POP3 Explicación: Este programa conceptualiza la recepción de correos electrónicos a través del protocolo POP3. El usuario introduce el servidor, usuario y contraseña. Al hacer clic en "Recibir", se conecta al servidor, autentica y lista los asuntos de los correos en un `ListBox`. (Requiere la librería OpenPop.NET).

Programa 189. Cliente de correo con conexión IMAP Explicación: Este programa conceptualiza la conexión a un servidor de correo IMAP. El usuario introduce los detalles del servidor, usuario y contraseña. Al hacer clic en "Recibir", se conecta al IMAP, autentica, abre la bandeja de entrada y lista los asuntos de los correos en un `ListBox`. (Requiere la librería MailKit).

Programa 190. Cliente de correo con adjuntos Explicación: Este cliente de correo permite adjuntar archivos a los mensajes. El usuario puede "Adjuntar Archivo" a través de un `OpenFileDialog`, y los nombres de los archivos se muestran en un `ListBox`. Al "Enviar", los archivos adjuntos se agregan al `MailMessage` antes de la preparación para el envío (simulado).

Programa 200. Cliente de correo con múltiples cuentas Explicación: Este programa simula un cliente de correo con soporte para múltiples cuentas de envío, seleccionables mediante un `ComboBox`. El usuario puede redactar un correo, y al hacer clic en "Enviar", se muestra un mensaje indicando que el correo sería enviado desde la cuenta seleccionada.

Programa 201. Cliente de correo con filtros de mensajes Explicación: Este programa permite aplicar filtros a los mensajes de correo. El usuario introduce un texto en un `TextBox` y al "Aplicar Filtro", el programa itera sobre los elementos de un `ListBox` (que simulan correos) y muestra un `MessageBox` para aquellos que contienen el texto del filtro.

Programa 202. Cliente de correo con carpetas personalizadas Explicación: Este programa simula la gestión de carpetas personalizadas para correos. El usuario puede introducir un nombre de carpeta en un `TextBox` y, al hacer clic en "Crear Carpeta", la nueva carpeta se añade a un `TreeView`.

Programa 203. Cliente de correo con notificaciones Explicación: Este programa utiliza un `NotifyIcon` para mostrar notificaciones de nuevos correos. Al hacer clic en "Notificar", se muestra una burbuja de notificación en la bandeja del sistema con el mensaje "Tienes un nuevo mensaje".

Programa 204. Cliente de correo con contactos Explicación: Este programa permite al usuario agregar contactos a una lista. Se introduce el nombre y el correo electrónico del contacto en `TextBoxes`, y al hacer clic en "Agregar Contacto", la información se añade a un `ListBox` en un formato legible.

Programa 205. Cliente de correo con calendario Explicación: Este programa integra un calendario (`MonthCalendar`) en el cliente de correo. Al hacer clic en "Seleccionar Fecha", la fecha elegida en el calendario se muestra en un `TextBox`, lo que podría usarse para programar o asociar correos con fechas.

Programa 206. Cliente de correo con recordatorios Explicación: Este programa permite al usuario agregar recordatorios asociados a un mensaje y una fecha/hora específica. La información del recordatorio (fecha/hora y mensaje) se añade y se muestra en un `ListBox`.

Programa 207. Cliente de correo con sincronización Explicación: Este programa conceptualiza la funcionalidad de sincronización de correos. Al hacer clic en "Sincronizar", se muestra un mensaje indicando que se ha iniciado la sincronización, aunque la implementación real requeriría una conexión a un servidor.

Programa 208. Cliente de correo con cifrado Explicación: Este programa permite cifrar el cuerpo de un mensaje de correo electrónico utilizando una codificación simple Base64. El usuario escribe el mensaje y, al hacer clic en "Cifrar", el texto cifrado se muestra en otro `TextBox`.

Programa 209. Cliente de correo con firmas personalizadas Explicación: Este programa permite añadir una firma personalizada al final del cuerpo del correo. El usuario escribe el cuerpo del mensaje y su firma por separado. Al hacer clic en "Enviar", se muestra un `MessageBox` con el mensaje completo, incluyendo la firma.

Programa 210. Sistema de login con logs de actividad Explicación: Este programa implementa un sistema de inicio de sesión que registra la actividad. Cuando un usuario inicia sesión con éxito (credenciales "admin"/"1234"), se registra la fecha, hora y nombre de usuario en un archivo de texto ("log_actividad.txt").

Programa 211. Sistema de login con auditoría Explicación: Este programa registra los intentos de inicio de sesión para fines de auditoría. Cada vez que un usuario intenta iniciar sesión, la fecha, hora y el nombre de usuario se añaden a un `ListBox`, mostrando un registro de todos los intentos, válidos o no.

Programa 212. Sistema de login con autenticación externa Explicación: Este programa conceptualiza la autenticación de usuarios contra un sistema externo. Simula una autenticación exitosa si las credenciales son "admin"/"1234", pero se muestra un mensaje indicando que la implementación real requeriría un servicio de autenticación externo.

Programa 213. Sistema de login con validación de IP Explicación: Este programa simula la validación de la dirección IP del usuario para el acceso. Se compara una IP actual (simulada) con una IP permitida. Si coinciden, se concede el acceso; de lo contrario, se deniega.

Programa 214. Sistema de login con tiempo de expiración Explicación: Este programa implementa una funcionalidad de tiempo de expiración de sesión. Al iniciar sesión, se establece un tiempo de expiración (ej. 30 minutos). Si se intenta acceder después de ese tiempo, se muestra un mensaje de "Tiempo de sesión expirado".

Programa 215. Sistema de login con recordatorio de sesión Explicación: Este programa incluye un `Timer` que, al iniciar sesión, se activa para mostrar un recordatorio periódico ("Recuerda cerrar sesión si terminas") en una etiqueta, incentivando al usuario a cerrar su sesión.

Programa 216. Sistema de login con restricción de horarios Explicación: Este programa implementa una restricción de horarios para el inicio de sesión. El acceso solo se permite dentro de un rango de tiempo específico (ej. de 8:00 a 18:00). Si se intenta iniciar sesión fuera de este horario, se deniega el acceso.

Programa 217. Sistema de login con acceso remoto Explicación: Este programa conceptualiza un sistema de inicio de sesión que verifica el acceso remoto. Al hacer clic en "Iniciar Sesión", se muestra un mensaje simulando que se ha verificado el acceso remoto, señalando que una implementación real requeriría un servidor.

Programa 218. Sistema de login con sincronización de datos Explicación: Este programa simula la sincronización de datos con un servidor al iniciar sesión. Al hacer clic en "Iniciar Sesión", se muestra un mensaje indicando que los datos se han sincronizado, aunque la funcionalidad subyacente de sincronización no está implementada.

Programa 219. Sistema de login con cifrado de base de datos Explicación: Este programa demuestra un cifrado simple para contraseñas antes de su hipotético almacenamiento en una base de datos. Al introducir una contraseña, se muestra su versión cifrada en Base64, ilustrando una forma básica de protección de datos.

Programa 220. Sistema de login con hash de contraseñas Explicación: Este programa muestra cómo generar un hash de una contraseña utilizando el algoritmo SHA256. Al introducir una contraseña, se calcula y se muestra su valor hash, lo cual es una práctica común para almacenar contraseñas de forma segura en bases de datos.

Programa 221. Sistema de inventario básico Explicación: Este programa es un sistema de inventario muy simple que permite agregar productos y sus cantidades a un `ListBox`. El usuario introduce el nombre del producto y la cantidad en `TextBoxes`, y al hacer clic en "Agregar", el registro se añade a la lista.

Programa 222. Sistema de inventario con base de datos Explicación: Este programa conceptualiza un sistema de inventario que utiliza una base de datos para almacenar productos y cantidades. Al cargar, se inicializa un `DataTable` como fuente de datos para un `DataGridView`. Al "Agregar", los nuevos productos se añaden al `DataTable`.

Programa 223. Sistema de inventario con facturación Explicación: Este programa permite agregar productos con sus cantidades y precios a una "factura" que se muestra en un `ListBox`. Al hacer clic en "Agregar", se calcula el total para ese producto y se añade el detalle a la lista de la factura.

Programa 224. Sistema de inventario con reportes Explicación: Este programa permite generar un reporte básico del inventario. Al hacer clic en "Generar Reporte", se compila una cadena de texto con todos los productos listados en el `ListBox` y se muestra en un `MessageBox`.

Programa 225. Sistema de inventario con gráficos Explicación: Este programa visualiza datos de inventario con un gráfico de columnas. Al hacer clic en "Mostrar Gráfico", se genera un gráfico de ejemplo (`Producto A`, `Producto B` con cantidades) en un `Chart`.

Programa 226. Sistema de inventario con exportación Excel Explicación: Este programa conceptualiza la exportación de datos de inventario a un archivo de Excel. Al hacer clic en "Exportar a Excel", se muestra un mensaje indicando que los datos se exportarían, requiriendo una referencia a la biblioteca de Microsoft Excel.

Programa 227. Sistema de inventario con importación Excel Explicación: Este programa conceptualiza la importación de datos de inventario desde un archivo de Excel. Al hacer clic en "Importar desde Excel", se muestra un mensaje indicando que los datos se importarían, requiriendo una referencia a la biblioteca de Microsoft Excel.

Programa 228. Sistema de inventario con códigos de barra Explicación: Este programa permite registrar códigos de barra en un `ListBox`. El usuario introduce un código en un `TextBox` y, al hacer clic en "Agregar Código", este se añade a la lista de códigos, simulando un registro de productos por código.

Programa 229. Sistema de inventario con múltiples usuarios Explicación: Este programa simula un sistema de inventario con roles de usuario (Admin, Vendedor) seleccionables en un `ComboBox`. Al agregar un producto, se muestra un mensaje indicando qué usuario agregó el producto y su cantidad, simulando la trazabilidad por usuario.

Programa 230. Sistema de inventario con permisos Explicación: Este programa simula un sistema de permisos basado en roles de usuario (Admin, Vendedor). Al intentar "Eliminar Producto", solo los usuarios con rol de "Admin" tendrían permiso; a los "Vendedores" se les denegaría la acción.

Programa 231. Sistema de inventario con auditoría Explicación: Este programa registra las acciones de adición de productos al inventario para fines de auditoría. Además de añadir el producto al `ListBox` de inventario, se registra la fecha, hora y el producto agregado en un `ListBox` de auditoría.

Programa 232. Sistema de inventario con estadísticas Explicación: Este programa permite mostrar estadísticas básicas del inventario. Al hacer clic en "Mostrar Estadísticas", se muestra un `MessageBox` con el número total de productos registrados en el `ListBox`.

Programa 233. Sistema de inventario con gráficos dinámicos Explicación: Este programa visualiza el inventario en un gráfico de columnas. Al hacer clic en "Actualizar Gráfico", se crea una serie donde cada elemento del `ListBox` de inventario se representa como un punto en el gráfico.

Programa 234. Sistema de inventario con múltiples almacenes Explicación: Este programa simula un sistema de inventario con la capacidad de gestionar productos en diferentes almacenes. El usuario selecciona un almacén de un `ComboBox` y luego agrega productos, los cuales se registran en el `ListBox` del inventario, indicando el almacén de origen.

Programa 235. Sistema de inventario con conexión remota Explicación: Este programa conceptualiza la conexión a un servidor remoto para gestionar el inventario. Al hacer clic en "Conectar a Servidor", se muestra un mensaje simulando una conexión remota establecida, indicando que la implementación real requeriría una infraestructura de servidor.

Programa 236. Sistema de inventario con backup automático Explicación: Este programa simula la función de realizar un backup automático de los datos del inventario. Al hacer clic en "Realizar Backup", se muestra un mensaje indicando que el backup se ha realizado correctamente.

Programa 237. Sistema de inventario con restauración de datos Explicación: Este programa simula la función de restaurar datos de inventario desde un backup previo. Al hacer clic en "Restaurar Backup", se muestra un mensaje indicando que los datos han sido restaurados.

Programa 238. Sistema de inventario con notificaciones Explicación: Este programa utiliza un `NotifyIcon` para alertar sobre stock bajo. Al hacer clic en "Notificar Stock Bajo", se muestra una burbuja de notificación en la bandeja del sistema con una advertencia sobre productos con stock bajo.

Programa 239. Sistema de inventario con control de lotes Explicación: Este programa permite añadir productos al inventario especificando también un número de lote. El usuario introduce el producto, el lote y la cantidad, y esta información se añade al `ListBox` del inventario.

Programa 240. Sistema de inventario con caducidades Explicación: Este programa permite gestionar productos en el inventario incluyendo su fecha de caducidad. El usuario introduce el producto, la cantidad y selecciona la fecha de caducidad con un `DateTimePicker`. La información se añade al `ListBox`.

Programa 241. Sistema de ventas básico Explicación: Este programa es un sistema de ventas muy simple que permite registrar ventas de productos y sus cantidades en un `ListBox`. El usuario introduce el nombre del producto y la cantidad, y al hacer clic en "Agregar Venta", el registro se añade a la lista.

Programa 242. Sistema de ventas con base de datos [280, 281 Venta", el registro en el `ListBox` incluye el nombre del cliente.

Programa 244. Sistema de ventas con facturación Explicación: Este programa permite generar ítems para una factura. El usuario introduce el producto, la cantidad y el precio. Al hacer clic en "Agregar Factura", se calcula el total para ese ítem y se añade a un `ListBox` que simula la factura.

Programa 245. Sistema de ventas con reportes Explicación: Este programa permite generar un reporte básico de las ventas registradas. Al hacer clic en "Generar Reporte", se compila una cadena de texto con todos los ítems de venta del `ListBox` y se muestra en un `MessageBox`.

Programa 246. Sistema de ventas con estadísticas Explicación: Este programa proporciona una función para mostrar estadísticas básicas de ventas. Al hacer clic en "Mostrar Estadísticas", se muestra un `MessageBox` con el número total de ventas registradas en el `ListBox`.

Programa 247. Sistema de ventas con gráficos Explicación: Este programa visualiza datos de ventas con un gráfico de columnas. Al hacer clic en "Mostrar Gráfico", se crea una serie donde cada ítem del `ListBox` de ventas se representa como un punto en el gráfico.

Programa 248. Sistema de ventas con exportación Excel Explicación: Este programa conceptualiza la exportación de datos de ventas a un archivo de Excel. Al hacer clic en "Exportar a Excel", se muestra un mensaje indicando que las ventas se exportarían, requiriendo una referencia a la biblioteca de Microsoft Excel.

Programa 249. Sistema de ventas con importación Excel Explicación: Este programa conceptualiza la importación de datos de ventas desde un archivo de Excel. Al hacer clic en "Importar desde Excel", se muestra un mensaje indicando que las ventas se importarían, requiriendo una referencia a la biblioteca de Microsoft Excel.

Programa 250. Sistema de ventas con múltiples usuarios Explicación: Este programa simula un sistema de ventas con roles de usuario (Admin, Vendedor) seleccionables en un `ComboBox`. Al agregar una venta, se muestra un mensaje indicando qué usuario realizó la venta y qué producto y cantidad fueron.

Programa 251. Sistema de ventas con permisos Explicación: Este programa simula un sistema de permisos basado en roles de usuario (Admin, Vendedor). Al intentar "Eliminar Venta", solo los usuarios con rol de "Admin" tendrían permiso; a los "Vendedores" se les denegaría la acción.

Programa 252. Sistema de ventas con auditoría Explicación: Este programa registra las acciones de adición de ventas para fines de auditoría. Además de añadir la venta al `ListBox` de ventas, se registra la fecha, hora y el producto vendido en un `ListBox` de auditoría.

Programa 253. Sistema de ventas con conexión remota [290, 2 backup de ventas se ha realizado correctamente.

Programa 255. Sistema de ventas con restauración de datos Explicación: Este programa simula la función de restaurar datos de ventas desde un backup previo. Al hacer clic en "Restaurar Backup", se muestra un mensaje indicando que los datos de ventas han sido restaurados.

Programa 256. Sistema de ventas con notificaciones Explicación: Este programa utiliza un `NotifyIcon` para notificar sobre nuevas ventas. Al hacer clic en "Notificar Venta", se muestra una burbuja de notificación en la bandeja del sistema con el mensaje "Nueva venta registrada".

Programa 257. Sistema de ventas con control de stock Explicación: Este programa simula un sistema de ventas que controla el stock de productos. Al intentar agregar una venta, se verifica si hay suficiente stock del producto. Si lo hay, se reduce el stock y se confirma la venta; de lo contrario, se notifica "Stock insuficiente".

Programa 258. Sistema de ventas con caducidades Explicación: Este programa permite registrar ventas de productos incluyendo su fecha de caducidad. El usuario introduce el producto, la cantidad y selecciona la fecha de caducidad con un `DateTimePicker`. La información se añade al `ListBox` de ventas.

Programa 259. Sistema de ventas con código de barras Explicación: Este programa permite registrar ventas introduciendo códigos de barras. El usuario escanea o introduce un código en un `TextBox` y, al hacer clic en "Agregar Venta", el código se añade a un `ListBox` de ventas.

Programa 260. Sistema de ventas con descuentos Explicación: Este programa permite aplicar descuentos a los productos en una venta. El usuario introduce el producto, su precio y un porcentaje de descuento. Al "Agregar Venta", se calcula el precio final con el descuento aplicado y se muestra en el `ListBox`.

Programa 261. Sistema escolar básico Explicación: Este programa es un sistema escolar muy simple que permite registrar alumnos y sus materias en un `ListBox`. El usuario introduce el nombre del alumno y la materia, y al hacer clic en "Agregar Registro", la información se añade a la lista.

Programa 262. Sistema escolar con base de datos Explicación: Este programa conceptualiza un sistema escolar que utiliza una base de datos para almacenar registros de alumnos y materias. Al cargar, se inicializa un** Explicación: Este programa permite registrar nuevos profesores en el sistema escolar. El usuario introduce el nombre del profesor y su materia en `TextBoxes`, y al hacer clic en "Agregar Profesor", la información se añade a un `ListBox` de profesores.

Programa 265. Sistema escolar con registro de materias Explicación: Este programa permite registrar nuevas materias en el sistema escolar. El usuario introduce el nombre de la materia en un `TextBox` y, al hacer clic en "Agregar Materia", el nombre se añade a un `ListBox` de materias.

Programa 266. Sistema escolar con registro de calificaciones Explicación: Este programa permite registrar calificaciones de alumnos por materia. El usuario introduce el nombre del alumno, la materia y la calificación. Al "Agregar Calificación", la información se añade a un `ListBox` de calificaciones.

Programa 267. Sistema escolar con reportes Explicación: Este programa permite generar un reporte básico de las calificaciones registradas. Al hacer clic en "Generar Reporte", se compila una cadena de texto con todos los registros de calificaciones del `ListBox` y se muestra en un `MessageBox`.

Programa 268. Sistema escolar con gráficos Explicación: Este programa visualiza datos de calificaciones con un gráfico de columnas. Al hacer clic en "Mostrar Gráfico", se crea una serie donde cada registro del `ListBox` de calificaciones se representa como un punto en el gráfico.

Programa 269. Sistema escolar con exportación Excel Explicación: Este programa conceptualiza la exportación de datos de calificaciones a un archivo de Excel. Al hacer clic en "Exportar a Excel", se muestra un mensaje indicando que las calificaciones se exportarían, requiriendo una referencia a la biblioteca de Microsoft Excel.

Programa 270. Sistema escolar con importación Excel Explicación: Este programa conceptualiza la importación de datos de calificaciones desde un archivo de Excel. Al permisos** Explicación: Este programa simula un sistema de permisos basado en roles de usuario (Admin, Profesor). Al intentar "Eliminar Registro", solo los usuarios con rol de "Admin" tendrían permiso; a los "Profesores" se les denegaría la acción.

Programa 273. Sistema escolar con auditoría Explicación: Este programa registra las acciones de adición de alumnos para fines de auditoría. Además de añadir el alumno, se registra la fecha, hora y el nombre del alumno agregado en un `ListBox` de auditoría.

Programa 274. Sistema escolar con conexión remota Explicación: Este programa conceptualiza la conexión a un servidor remoto para gestionar el sistema escolar. Al hacer clic en "Conectar a Servidor", se muestra un mensaje simulando una conexión remota establecida, indicando que la implementación real requeriría una infraestructura de servidor.

Programa 275. Sistema escolar con backup automático notificaciones** Explicación: Este programa utiliza un `NotifyIcon` para notificar eventos en el sistema escolar. Al hacer clic en "Notificar Evento", se muestra una burbuja de notificación en la bandeja del sistema con el mensaje "Nuevo evento registrado".

Programa 278. Sistema escolar con estadísticas Explicación: Este programa proporciona una función para mostrar estadísticas básicas del sistema escolar. Al hacer clic en "Mostrar Estadísticas", se muestra un `MessageBox` con el número total de registros (asumiendo que los ítems en `lstCalificaciones` representan registros de alumnos o similar).

Programa 279. Sistema escolar con boletines Explicación: Este programa permite generar un boletín de calificaciones simple. Al hacer clic en "Generar Boletín", se compila una cadena de texto con todos los registros de calificaciones del `ListBox` y se muestra en un `MessageBox` como un boletín.

**Programa 280. Sistema escolar con login de usuarios El usuario introduce el nombre del paciente y el doctor, y al hacer clic en "Agregar Registro", la información se añade a la lista.

Programa 282. Sistema médico con registro de pacientes Explicación: Este programa permite registrar nuevos pacientes en el sistema médico. El usuario introduce el nombre y la edad del paciente en `TextBoxes`, y al hacer clic en "Agregar Paciente", la información se añade a un `ListBox` de pacientes.

Programa 283. Sistema médico con registro de doctores Explicación: Este programa permite registrar nuevos doctores en el sistema médico. El usuario introduce el nombre del doctor y su especialidad en `TextBoxes`, y al hacer clic en "Agregar Doctor", la información se añade a un `ListBox` de doctores.

Programa 284. Sistema médico con registro de citas Explicación: Este programa permite registrar citas médicas. El usuario introduce el paciente, el doctor y selecciona la fecha de la cita con un `DateTimePicker`. Al "Agregar Cita", la información se añade a un `ListBox` la factura. Al "Generar Factura", la información se añade a un `ListBox` de facturas.

Programa 287. Sistema médico con reportes Explicación: Este programa permite generar un reporte básico del historial clínico. Al hacer clic en "Generar Reporte", se compila una cadena de texto con todos los registros del historial del `ListBox` y se muestra en un `MessageBox`.

Programa 288. Sistema médico con gráficos Explicación: Este programa visualiza datos de citas médicas con un gráfico de columnas. Al hacer clic en "Mostrar Gráfico", se crea una serie donde cada registro del `ListBox` de citas se representa como un punto en el gráfico.

Programa 289. Sistema médico con exportación Excel Explicación: Este programa conceptualiza la exportación de datos médicos a un archivo de Excel. Al hacer clic en "Exportar a Excel", se muestra un mensaje indicando que los datos se exportarían, requiriendo una referencia a la biblioteca de Microsoft Excel.

Programa 290. Sistema médico con importación Excel Explicación: Este programa conceptualiza la importación de datos médicos desde un archivo de Excel. Al hacer clic en "Importar desde Excel", se muestra un mensaje indicando que los datos se importarían, requiriendo una referencia a la biblioteca de Microsoft Excel.

Programa 291. Sistema médico con múltiples usuarios Explicación: Este programa simula un 293. Sistema médico con auditoría** Explicación: Este programa registra las acciones de adición de pacientes para fines de auditoría. Además de añadir el paciente, se registra la fecha, hora y el nombre del paciente agregado en un `ListBox` de auditoría.

Programa 294. Sistema médico con conexión remota Explicación: Este programa conceptualiza la conexión a un servidor remoto para gestionar el sistema médico. Al hacer clic en "Conectar a Servidor", se muestra un mensaje simulando una conexión remota establecida, indicando que la implementación real requeriría una infraestructura de servidor.

Programa 295. Sistema médico con backup automático Explicación: Este programa simula la función de realizar un backup automático de los datos médicos. Al hacer clic en "Realizar Backup", se muestra un mensaje indicando que el backup de datos médicos se ha realizado correctamente.

Programa 296. Sistema médico con restauración de datos 8].

Programa 298. Sistema médico con recordatorios de citas Explicación: Este programa permite programar y recordar citas médicas. Al hacer clic en "Recordar Cita", se muestra un `MessageBox` con un recordatorio de la cita programada para la fecha seleccionada en el `DateTimePicker`.

Programa 299. Sistema médico con estadísticas Explicación: Este programa proporciona una función para mostrar estadísticas básicas del sistema médico. Al hacer clic en "Mostrar Estadísticas", se muestra un `MessageBox` con el número total de pacientes registrados en el `ListBox`.

Programa 300. Sistema médico con login de usuarios Explicación: Este programa implementa un sistema de inicio de sesión para el sistema médico. El usuario introduce un nombre de usuario y una contraseña. Si las credenciales ("admin"/"1234") son correctas, se muestra "Login exitoso"; de lo contrario, "Usuario o contraseña incorrectos".

Manual

Manual de Ejercicios en VB.NET FORM

Programa 1. Formulario con un botón que muestra un mensaje

- **¿En qué consiste el ejercicio?** Este programa es una introducción básica a la programación de interfaces de usuario, demostrando cómo mostrar un simple cuadro de mensaje (MessageBox) en pantalla al hacer clic en un botón.
- **Controles necesarios en el formulario y sus propiedades:**
    - **Button**:
        - **Name**: `btnMensaje`.
            - **Propiedad**: Identifica de forma única al control `Button` dentro del código del formulario, permitiendo referenciarlo y asociar eventos a él.
            - **Utilización**: Se usa para que el usuario interactúe con la aplicación, en este caso, al hacer clic, desencadena una acción.
        - **Text**: "Mostrar Mensaje".
            - **Propiedad**: Define el texto visible que aparece en la superficie del botón. Este es el texto que el usuario lee en la interfaz.
            - **Utilización**: Proporciona una indicación clara al usuario sobre la acción que realizará el botón al ser presionado.
- **Utilización desde el programa:**
    - El código se ejecuta dentro de la clase `Form1`.
    - Cuando el usuario hace clic en el botón `btnMensaje`, se activa el evento `btnMensaje_Click`.
    - Dentro de este evento, la línea `MessageBox.Show("Hola, este es un mensaje")` despliega una ventana emergente que muestra el texto "Hola, este es un mensaje". La clase `MessageBox` se utiliza para mostrar cuadros de diálogo simples con mensajes al usuario.

Programa 2. Formulario con una etiqueta y un botón que cambia el texto

- **¿En qué consiste el ejercicio?** Este ejercicio ilustra cómo un botón puede interactuar con otros controles para modificar sus propiedades en tiempo de ejecución. En este caso, el programa cambia el texto de una etiqueta (Label) cuando el usuario hace clic en el botón.
- **Controles necesarios en el formulario y sus propiedades:**
    - **Label**:
        - **Name**: `lblTexto`.
            - **Propiedad**: Identificador único de la etiqueta en el código.

        - **Utilización**: Permite al programa acceder y modificar las propiedades de esta etiqueta específica.
    - **Text**: "Texto inicial".
        - **Propiedad**: Contiene el texto visible que la etiqueta muestra al usuario. Este texto puede ser modificado programáticamente.
        - **Utilización**: Se usa para mostrar información al usuario, y en este programa, su contenido se actualiza para reflejar un cambio.
- **Button**:
    - **Name**: `btnCambiar`.
        - **Propiedad**: Identificador único del botón.
        - **Utilización**: Permite asociar el evento de clic a este botón.
    - **Text**: "Cambiar Texto".
        - **Propiedad**: Texto visible del botón.
        - **Utilización**: Indica la acción que el botón realizará al ser presionado.
- **Utilización desde el programa:**
    - Cuando el usuario hace clic en el botón `btnCambiar`, se ejecuta el evento `btnCambiar_Click`.
    - La línea `lblTexto.Text = "Texto cambiado"` actualiza la propiedad `Text` del control `lblTexto` (la etiqueta) a "Texto cambiado". Esto provoca que el texto visible en el formulario cambie, mostrando la interacción entre los controles.

Programa 3. Formulario con un cuadro de texto y un botón para mostrar su contenido

- **¿En qué consiste el ejercicio?** Este programa enseña cómo capturar la entrada del usuario desde un control `TextBox` y luego mostrar ese contenido en un `MessageBox`. Es fundamental para comprender la interacción del usuario con la aplicación.
- **Controles necesarios en el formulario y sus propiedades:**
    - **TextBox**:
        - **Name**: `txtEntrada`.
            - **Propiedad**: Identificador único del cuadro de texto.
            - **Utilización**: Permite al usuario escribir texto. El programa puede leer el contenido de esta propiedad para procesar la entrada del usuario.
    - **Button**:
        - **Name**: `btnMostrar`.
            - **Propiedad**: Identificador único del botón.
            - **Utilización**: Se usa para que el usuario desencadene la acción de mostrar el contenido del `TextBox`.
        - **Text**: "Mostrar".
            - **Propiedad**: Texto visible del botón.
            - **Utilización**: Indica la acción que realizará el botón.
- **Utilización desde el programa:**

- Al hacer clic en el botón `btnMostrar`, se dispara el evento `btnMostrar_Click`.
- Dentro de este evento, la línea `MessageBox.Show("Contenido: " & txtEntrada.Text)` realiza dos acciones:
    - Toma el texto actual del `TextBox` `txtEntrada` utilizando su propiedad `txtEntrada.Text`.
    - Concatena este texto con la cadena "Contenido: " (usando el operador &) y muestra el resultado combinado en un cuadro de mensaje emergente. Esto demuestra cómo capturar y presentar la entrada del usuario.

Programa 4. Formulario con dos botones que cambian el color de fondo

- **¿En qué consiste el ejercicio?** Este ejercicio demuestra cómo modificar las propiedades visuales del formulario, específicamente su color de fondo, mediante la interacción con dos botones diferentes, cada uno asignando un color distinto.
- **Controles necesarios en el formulario y sus propiedades:**
    - **Button**:
        - **Name**: `btnRojo`.
            - **Propiedad**: Identificador único del botón para cambiar el color a rojo.
            - **Utilización**: Al ser clicado, su evento asociado cambia el `BackColor` del formulario.
        - **Text**: "Fondo Rojo".
            - **Propiedad**: Texto visible del botón.
            - **Utilización**: Informa al usuario que este botón cambiará el fondo a rojo.
    - **Button**:
        - **Name**: `btnAzul`.
            - **Propiedad**: Identificador único del botón para cambiar el color a azul.
            - **Utilización**: Al ser clicado, su evento asociado cambia el `BackColor` del formulario.
        - **Text**: "Fondo Azul".
            - **Propiedad**: Texto visible del botón.
            - **Utilización**: Informa al usuario que este botón cambiará el fondo a azul.
    - **Formulario (Me)**:
        - **BackColor**: Se refiere a la propiedad del color de fondo del formulario mismo.
            - **Propiedad**: Determina el color del fondo del área del formulario.
            - **Utilización**: Se utiliza para cambiar dinámicamente el aspecto visual del formulario.
- **Utilización desde el programa:**

- Cuando se hace clic en `btnRojo`, el evento `btnRojo_Click` se activa. Dentro de este evento, `Me.BackColor = Color.Red` cambia el color de fondo del formulario (representado por la palabra clave `Me`) a rojo.
- De manera similar, cuando se hace clic en `btnAzul`, el evento `btnAzul_Click` cambia el color de fondo del formulario a azul con `Me.BackColor = Color.Blue`.
- `Color.Red` y `Color.Blue` son constantes predefinidas de la clase `Color` que representan colores específicos.

Programa 5. Formulario con un cuadro de texto que convierte a mayúsculas

- **¿En qué consiste el ejercicio?** Este programa demuestra la manipulación de cadenas de texto ingresadas por el usuario, convirtiéndolas a mayúsculas directamente en el mismo `TextBox` donde fueron introducidas.
- **Controles necesarios en el formulario y sus propiedades:**
  - **TextBox**:
    - **Name**: `txtEntrada`.
      - **Propiedad**: Identificador único del cuadro de texto.
      - **Utilización**: Permite al usuario escribir cualquier texto. El contenido de esta propiedad es leído, transformado y luego actualizado.
  - **Button**:
    - **Name**: `btnMayusculas`.
      - **Propiedad**: Identificador único del botón.
      - **Utilización**: Se usa para que el usuario inicie la acción de convertir el texto a mayúsculas.
    - **Text**: "Convertir".
      - **Propiedad**: Texto visible del botón.
      - **Utilización**: Indica la función del botón.
- **Utilización desde el programa:**
  - Al hacer clic en el botón `btnMayusculas`, se ejecuta el evento `btnMayusculas_Click`.
  - Dentro de este evento, la línea `txtEntrada.Text = txtEntrada.Text.ToUpper()` realiza la conversión:
    - `txtEntrada.Text.ToUpper()`: Llama al método `ToUpper()` de la cadena de texto actual en el `TextBox`. Este método devuelve una nueva cadena con todos los caracteres convertidos a mayúsculas.
    - `txtEntrada.Text = ...`: Asigna esta nueva cadena (ahora en mayúsculas) de vuelta a la propiedad `Text` del `TextBox`, actualizando lo que el usuario ve en la interfaz.

Programa 6. Formulario con dos cuadros de texto para sumar números

- **¿En qué consiste el ejercicio?** Este programa introduce la captura de múltiples entradas numéricas de diferentes `TextBox`s, su conversión a valores numéricos, la realización de una operación aritmética básica (suma), y finalmente, la muestra del resultado en un `Label`.
- **Controles necesarios en el formulario y sus propiedades:**
    - **TextBox**:
        - **Name**: `txtNum1`.
            - **Propiedad**: Identificador único del primer cuadro de texto.
            - **Utilización**: Permite al usuario ingresar el primer número para la operación de suma.
    - **TextBox**:
        - **Name**: `txtNum2`.
            - **Propiedad**: Identificador único del segundo cuadro de texto.
            - **Utilización**: Permite al usuario ingresar el segundo número para la operación de suma.
    - **Button**:
        - **Name**: `btnSumar`.
            - **Propiedad**: Identificador único del botón.
            - **Utilización**: Desencadena el proceso de lectura, suma y visualización del resultado.
        - **Text**: "Sumar".
            - **Propiedad**: Texto visible del botón.
            - **Utilización**: Indica la acción principal del botón.
    - **Label**:
        - **Name**: `lblResultado`.
            - **Propiedad**: Identificador único de la etiqueta.
            - **Utilización**: Muestra el resultado de la operación. Su `Text` se actualiza con el cálculo.
        - **Text**: "Resultado:".
            - **Propiedad**: Texto inicial de la etiqueta.
- **Utilización desde el programa:**
    - Cuando se hace clic en el botón `btnSumar`, se activa el evento `btnSumar_Click`.
    - Las líneas `Dim a As Double = Val(txtNum1.Text)` y `Dim b As Double = Val(txtNum2.Text)` obtienen el texto de cada `TextBox` y utilizan la función `Val()` para convertirlos a valores numéricos de tipo `Double`. Esto es crucial porque el texto de los `TextBox` es una cadena, no un número.
    - Estos valores numéricos (`a` y `b`) se suman (`a + b`).
    - Finalmente, `lblResultado.Text = "Resultado: " & (a + b).ToString()` convierte el resultado de la suma a una cadena (usando `.ToString()`) y lo concatena con "Resultado: " para mostrarlo en la etiqueta `lblResultado`.

Programa 7. Formulario con un botón para limpiar cuadros de texto

- **¿En qué consiste el ejercicio?** Este programa enseña una funcionalidad común en aplicaciones: cómo limpiar el contenido de uno o varios cuadros de texto a la vez con la interacción de un solo botón.
- **Controles necesarios en el formulario y sus propiedades:**
    - **TextBox**:
        - **Name**: `txtUno`.
            - **Propiedad**: Identificador único del primer cuadro de texto.
            - **Utilización**: Almacena texto que el usuario puede ingresar, y su contenido puede ser borrado.
    - **TextBox**:
        - **Name**: `txtDos`.
            - **Propiedad**: Identificador único del segundo cuadro de texto.
            - **Utilización**: Similar a `txtUno`, permite la entrada de texto y puede ser vaciado.
    - **Button**:
        - **Name**: `btnLimpiar`.
            - **Propiedad**: Identificador único del botón.
            - **Utilización**: Desencadena la acción de limpiar los cuadros de texto.
        - **Text**: "Limpiar".
            - **Propiedad**: Texto visible del botón.
            - **Utilización**: Indica claramente la función del botón.
- **Utilización desde el programa:**
    - Al hacer clic en el botón `btnLimpiar`, se ejecuta el evento `btnLimpiar_Click`.
    - Las líneas `txtUno.Clear()` y `txtDos.Clear()` llaman al método `Clear()` de cada `TextBox` respectivamente. El método `Clear()` es una forma sencilla de eliminar todo el texto de un `TextBox`, dejándolo vacío. Esto es útil para restablecer los campos de entrada después de una operación.

Programa 8. Formulario con un botón que cierra la aplicación

- **¿En qué consiste el ejercicio?** Este es un programa muy simple pero esencial que muestra cómo terminar la ejecución de una aplicación de manera programática mediante un botón.
- **Controles necesarios en el formulario y sus propiedades:**
    - **Button**:
        - **Name**: `btnSalir`.
            - **Propiedad**: Identificador único del botón.
            - **Utilización**: Es el control que el usuario presiona para salir de la aplicación.
        - **Text**: "Salir".
            - **Propiedad**: Texto visible del botón.

        - **Utilización**: Indica claramente la función de cerrar la aplicación.
- **Utilización desde el programa:**
    - Cuando el usuario hace clic en el botón `btnSalir`, se activa el evento `btnSalir_Click`.
    - Dentro de este evento, la línea `Application.Exit()` detiene todos los mensajes y finaliza la ejecución de la aplicación, cerrando el formulario y terminando el programa. Es el método estándar para cerrar una aplicación de Windows Forms.

Programa 9. Formulario con una etiqueta que muestra la hora actual

- **¿En qué consiste el ejercicio?** Este programa introduce el uso del control `Timer` para actualizar periódicamente un `Label` con la hora actual del sistema, demostrando un concepto de interactividad en tiempo real y la ejecución de código a intervalos regulares.
- **Controles necesarios en el formulario y sus propiedades:**
    - **Label**:
        - **Name**: `lblHora`.
            - **Propiedad**: Identificador único de la etiqueta.
            - **Utilización**: Se usa para mostrar la hora actual. Su texto se actualiza cada segundo.
        - **Text**: "Hora:".
            - **Propiedad**: Texto inicial que muestra la etiqueta.
    - **Timer**:
        - **Name**: `Timer1`.
            - **Propiedad**: Identificador único del control `Timer`.
            - **Utilización**: Es el componente que genera eventos a intervalos regulares.
        - **Enabled**: `True`.
            - **Propiedad**: Una propiedad booleana que, cuando se establece en `True`, hace que el `Timer` comience a funcionar y dispare su evento `Tick`.
            - **Utilización**: Asegura que el `Timer` esté activo desde el inicio de la aplicación.
        - **Interval**: `1000` (milisegundos).
            - **Propiedad**: Define el tiempo en milisegundos que debe transcurrir entre cada disparo del evento `Tick` del `Timer`.
            - **Utilización**: Un valor de `1000` significa que el evento `Tick` se disparará cada 1 segundo, permitiendo una actualización constante de la hora.
- **Utilización desde el programa:**
    - El `Timer1` está habilitado (`Enabled: True`) y configurado para disparar su evento `Tick` cada 1000 milisegundos (1 segundo).

- En cada ocurrencia del evento `Timer1_Tick`, la línea `lblHora.Text = "Hora: " & DateTime.Now.ToLongTimeString()` se ejecuta:
    - `DateTime.Now.ToLongTimeString()`: Obtiene la fecha y hora actuales del sistema (`DateTime.Now`) y las convierte a una cadena de texto que representa solo la hora en un formato largo (ej., "10:30:45 AM").
    - Esta cadena de hora se concatena con "Hora: " y se asigna a `lblHora.Text`, actualizando el texto visible de la etiqueta cada segundo.

Programa 10. Formulario con un cuadro de texto que cuenta caracteres

- **¿En qué consiste el ejercicio?** Este ejercicio demuestra cómo reaccionar a los cambios en el texto de un `TextBox` en tiempo real y mostrar información dinámica, como el número de caracteres que el usuario ha ingresado.
- **Controles necesarios en el formulario y sus propiedades:**
    - **TextBox**:
        - **Name**: `txtEntrada`.
            - **Propiedad**: Identificador único del cuadro de texto.
            - **Utilización**: Permite al usuario introducir texto. El programa monitorea los cambios en su contenido para actualizar el contador.
        - **Multiline**: `True`.
            - **Propiedad**: Si se establece en `True`, permite que el `TextBox` ocupe varias líneas y, opcionalmente, muestre barras de desplazamiento si el texto excede el tamaño visible.
            - **Utilización**: Habilita la entrada de texto más extenso, como un párrafo.
    - **Label**:
        - **Name**: `lblContar`.
            - **Propiedad**: Identificador único de la etiqueta.
            - **Utilización**: Se usa para mostrar el número actual de caracteres en el `TextBox`. Su texto se actualiza dinámicamente.
        - **Text**: "Caracteres: 0".
            - **Propiedad**: Texto inicial de la etiqueta.
- **Utilización desde el programa:**
    - El evento `txtEntrada_TextChanged` se dispara automáticamente cada vez que el texto dentro del `TextBox` `txtEntrada` cambia (por ejemplo, cuando el usuario escribe o borra un carácter).
    - Dentro de este evento, la línea `lblContar.Text = "Caracteres: " & txtEntrada.Text.Length.ToString()` realiza la actualización:
        - `txtEntrada.Text.Length`: Obtiene la cantidad de caracteres en la cadena de texto actual del `TextBox`. `Length` es una propiedad de las cadenas que devuelve su longitud.

- `.ToString()`: Convierte el valor numérico de la longitud a una cadena de texto.
- Esta cadena se concatena con "Caracteres: " y se asigna a `lblContar.Text`, actualizando el contador visible en la etiqueta en tiempo real.

Programa 11. Formulario con dos botones para mostrar imágenes diferentes

- **¿En qué consiste el ejercicio?** Este programa enseña cómo cargar y mostrar imágenes desde archivos locales en un control `PictureBox` en una aplicación. Permite al usuario alternar entre dos imágenes diferentes utilizando botones.
- **Controles necesarios en el formulario y sus propiedades:**
    - **PictureBox**:
        - **Name**: `picImagen`.
            - **Propiedad**: Identificador único del control `PictureBox`.
            - **Utilización**: Es el contenedor donde se visualizarán las imágenes. Su propiedad `Image` se actualiza para mostrar diferentes archivos.
        - **SizeMode**: `StretchImage`.
            - **Propiedad**: Controla cómo se ajusta la imagen dentro de los límites del `PictureBox`. `StretchImage` redimensiona la imagen (estirándola o encogiéndola) para que encaje perfectamente en el tamaño del control.
            - **Utilización**: Asegura que las imágenes, independientemente de su tamaño original, se muestren completamente dentro del `PictureBox`.
        - **Size**: `200x200`.
            - **Propiedad**: Establece las dimensiones (ancho y alto) del control `PictureBox` en píxeles.
            - **Utilización**: Define el área en la que se mostrará la imagen.
    - **Button**:
        - **Name**: `btnImagen1`.
            - **Propiedad**: Identificador único del botón para cargar la primera imagen.
            - **Utilización**: Al hacer clic, carga y muestra el archivo "imagen1.jpg".
        - **Text**: "Imagen 1".
            - **Propiedad**: Texto visible del botón.
            - **Utilización**: Indica la imagen que cargará.
    - **Button**:
        - **Name**: `btnImagen2`.
            - **Propiedad**: Identificador único del botón para cargar la segunda imagen.
            - **Utilización**: Al hacer clic, carga y muestra el archivo "imagen2.jpg".
        - **Text**: "Imagen 2".

        - **Propiedad**: Texto visible del botón.
        - **Utilización**: Indica la imagen que cargará.
- **Utilización desde el programa:**
    - Cuando se hace clic en `btnImagen1`, se ejecuta el evento `btnImagen1_Click`. Dentro de este, `picImagen.Image = Image.FromFile("imagen1.jpg")` carga la imagen del archivo "imagen1.jpg" (asumiendo que está en la misma carpeta que el ejecutable o se especifica una ruta completa) y la asigna a la propiedad `Image` del `PictureBox`.
    - De forma similar, cuando se hace clic en `btnImagen2`, el evento `btnImagen2_Click` carga y muestra "imagen2.jpg".
    - `Image.FromFile()` es un método estático que crea un objeto `Image` a partir de un archivo de imagen, que luego puede ser asignado a un `PictureBox`.

Programa 12. Formulario con un botón que cambia el tamaño de la ventana

- **¿En qué consiste el ejercicio?** Este ejercicio demuestra cómo modificar las dimensiones (ancho y alto) del formulario principal de la aplicación programáticamente, es decir, mediante código, en respuesta a la interacción del usuario con un botón.
- **Controles necesarios en el formulario y sus propiedades:**
    - **Button**:
        - **Name**: `btnCambiarTamaño`.
            - **Propiedad**: Identificador único del botón.
            - **Utilización**: Es el control que el usuario presiona para desencadenar el cambio de tamaño del formulario.
        - **Text**: "Cambiar Tamaño".
            - **Propiedad**: Texto visible del botón.
            - **Utilización**: Indica claramente la acción que realizará el botón.
    - **Formulario (Me)**:
        - **Size**: Se refiere a la propiedad del formulario que contiene sus dimensiones (ancho y alto) en píxeles.
            - **Propiedad**: Es un objeto `Size` que encapsula el ancho y el alto del formulario.
            - **Utilización**: Se utiliza para leer o establecer las dimensiones del formulario en tiempo de ejecución.
- **Utilización desde el programa:**
    - Al hacer clic en el botón `btnCambiarTamaño`, el evento `btnCambiarTamaño_Click` se ejecuta.
    - La línea `Me.Size = New Size(800, 600)` realiza el cambio de tamaño:
        - `Me`: Hace referencia a la instancia actual del formulario.
        - `New Size(800, 600)`: Crea un nuevo objeto `Size` con un ancho de 800 píxeles y una altura de 600 píxeles.

- Este nuevo objeto `Size` se asigna a la propiedad `Me.Size`, estableciendo las nuevas dimensiones del formulario en pantalla.

Programa 13. Formulario con un cuadro de texto que solo permite números

- **¿En qué consiste el ejercicio?** Este programa enseña una técnica importante de validación de entrada en tiempo real: restringir un `TextBox` para que solo acepte caracteres numéricos. Esto evita que el usuario ingrese texto no numérico donde solo se esperan números.
- **Controles necesarios en el formulario y sus propiedades:**
    - **TextBox**:
        - **Name**: `txtNumeros`.
            - **Propiedad**: Identificador único del cuadro de texto.
            - **Utilización**: Es el campo de entrada donde el usuario intentará escribir. La lógica del programa intercepta las pulsaciones de teclas para filtrar solo los números.
- **Utilización desde el programa:**
    - El evento `txtNumeros_KeyPress` se dispara cada vez que una tecla es presionada mientras el `TextBox` `txtNumeros` tiene el foco. Este evento es ideal para la validación de caracteres individuales.
    - Dentro de este evento, la condición `If Not Char.IsControl(e.KeyChar) AndAlso Not Char.IsDigit(e.KeyChar) Then` verifica dos cosas:
        - `Char.IsControl(e.KeyChar)`: Verifica si el carácter de la tecla presionada (`e.KeyChar`) es una tecla de control (como `Backspace` para borrar, `Enter`, `Tab`, etc.). Estas teclas suelen permitirse para la edición normal del texto.
        - `Char.IsDigit(e.KeyChar)`: Verifica si el carácter de la tecla presionada es un dígito numérico (del 0 al 9).
    - La condición completa (`Not Char.IsControl(...) AndAlso Not Char.IsDigit(...)`) significa: "Si la tecla NO es un control Y NO es un dígito...".
    - Si esta condición es verdadera (es decir, el usuario presionó una tecla que no es un número ni una tecla de control), entonces `e.Handled = True` se establece.
        - **e.Handled**: Una propiedad booleana del objeto `KeyPressEventArgs`. Si se establece en `True`, indica que el evento ha sido completamente manejado por el código y el carácter asociado no debe ser procesado ni aparecer en el `TextBox`.
        - **Utilización**: Esto evita que caracteres no numéricos aparezcan en el `TextBox`, forzando la entrada solo a números.

Programa 14. Formulario con tres botones que cambian el color de texto

- **¿En qué consiste el ejercicio?** Este ejercicio demuestra cómo cambiar el color de la fuente de un control `Label` utilizando múltiples botones. Cada botón está configurado para asignar un color de fuente diferente a la misma etiqueta.
- **Controles necesarios en el formulario y sus propiedades:**
    - **Label**:
        - **Name**: `lblTexto`.
            - **Propiedad**: Identificador único de la etiqueta.
            - **Utilización**: Muestra un texto de ejemplo cuyo color de fuente será modificado por los botones.
        - **Text**: "Texto de ejemplo".
            - **Propiedad**: Contenido de texto inicial de la etiqueta.
    - **Button**:
        - **Name**: `btnRojo`.
            - **Propiedad**: Identificador único para el botón que cambia el texto a rojo.
            - **Utilización**: Al hacer clic, modifica el `ForeColor` de `lblTexto` a `Color.Red`.
        - **Text**: "Rojo".
            - **Propiedad**: Texto visible del botón.
    - **Button**:
        - **Name**: `btnVerde`.
            - **Propiedad**: Identificador único para el botón que cambia el texto a verde.
            - **Utilización**: Al hacer clic, modifica el `ForeColor` de `lblTexto` a `Color.Green`.
        - **Text**: "Verde".
            - **Propiedad**: Texto visible del botón.
    - **Button**:
        - **Name**: `btnAzul`.
            - **Propiedad**: Identificador único para el botón que cambia el texto a azul.
            - **Utilización**: Al hacer clic, modifica el `ForeColor` de `lblTexto` a `Color.Blue`.
        - **Text**: "Azul".
            - **Propiedad**: Texto visible del botón.
    - **Label.ForeColor**:
        - **Propiedad**: Define el color de la fuente del texto que se muestra en la etiqueta.
        - **Utilización**: Es la propiedad clave que se modifica para cambiar la apariencia del texto.
- **Utilización desde el programa:**

- Cada uno de los botones (`btnRojo`, `btnVerde`, `btnAzul`) tiene un evento `_Click` asociado que se activa cuando el usuario lo presiona.
- Dentro de cada evento, la propiedad `ForeColor` del `lblTexto` se establece en el color correspondiente:
    - `lblTexto.ForeColor = Color.Red` (para `btnRojo_Click`).
    - `lblTexto.ForeColor = Color.Green` (para `btnVerde_Click`).
    - `lblTexto.ForeColor = Color.Blue` (para `btnAzul_Click`).
- `Color.Red`, `Color.Green`, `Color.Blue` son constantes predefinidas de la clase `Color` que representan colores específicos.

Programa 15. Formulario con un cuadro de texto y un botón para invertir texto

- **¿En qué consiste el ejercicio?** Este programa introduce la manipulación de cadenas de texto para invertir su orden de caracteres. El usuario ingresa un texto en un `TextBox`, y al presionar un botón, el texto dentro del mismo `TextBox` se muestra al revés.
- **Controles necesarios en el formulario y sus propiedades:**
    - **TextBox**:
        - **Name**: `txtEntrada`.
            - **Propiedad**: Identificador único del cuadro de texto.
            - **Utilización**: Permite al usuario introducir el texto que desea invertir. El contenido de esta propiedad es leído y luego actualizado con el texto invertido.
    - **Button**:
        - **Name**: `btnInvertir`.
            - **Propiedad**: Identificador único del botón.
            - **Utilización**: Desencadena la acción de invertir el texto.
        - **Text**: "Invertir".
            - **Propiedad**: Texto visible del botón.
            - **Utilización**: Indica claramente la función del botón.
- **Utilización desde el programa:**
    - Al hacer clic en el botón `btnInvertir`, se ejecuta el evento `btnInvertir_Click`.
    - Dentro de este evento, el proceso es el siguiente:
        - `Dim texto As String = txtEntrada.Text`: Se declara una variable `texto` de tipo `String` y se le asigna el contenido actual del `TextBox`.
        - `Dim invertido As String = StrReverse(texto)`: Se declara una variable `invertido` y se le asigna el resultado de la función `StrReverse()`.
            - **StrReverse()**: Es una función de Visual Basic que toma una cadena de texto como argumento (en este caso, `texto`) y devuelve una *nueva* cadena con todos sus caracteres en orden inverso.

        - **Utilización**: Es la función clave para realizar la inversión del texto.
    - `txtEntrada.Text = invertido`: El texto invertido (`invertido`) se asigna de nuevo a la propiedad `Text` del `TextBox` `txtEntrada`, actualizando el contenido visible en el formulario.

Programa 16. Formulario con un contador de clics

- **¿En qué consiste el ejercicio?** Este programa demuestra cómo mantener y actualizar un estado (en este caso, un contador numérico) dentro del formulario a medida que el usuario interactúa con un botón. El contador persiste a lo largo de las interacciones del usuario.
- **Controles necesarios en el formulario y sus propiedades:**
    - **Label**:
        - **Name**: `lblContador`.
            - **Propiedad**: Identificador único de la etiqueta.
            - **Utilización**: Muestra el valor actual del contador de clics. Su texto se actualiza con cada clic.
        - **Text**: "Clics: 0".
            - **Propiedad**: Texto inicial que muestra la etiqueta, indicando que el contador comienza en cero.
    - **Button**:
        - **Name**: `btnContar`.
            - **Propiedad**: Identificador único del botón.
            - **Utilización**: Es el control que el usuario presiona para incrementar el contador.
        - **Text**: "Contar Clic".
            - **Propiedad**: Texto visible del botón.
            - **Utilización**: Indica la acción que el botón realizará.
- **Utilización desde el programa:**
    - **Dim contador As Integer = 0**: Se declara una variable llamada `contador` de tipo `Integer` (entero) y se inicializa a 0. Es crucial que esta variable se declare **a nivel de clase** (fuera de cualquier procedimiento de evento) para que su valor persista y no se reinicie cada vez que se hace clic en el botón.
    - Cada vez que se hace clic en el botón `btnContar`, el evento `btnContar_Click` se activa.
    - Dentro de este evento:
        - `contador += 1`: Esta línea incrementa el valor de la variable `contador` en 1. Es una forma abreviada de `contador = contador + 1`.
        - `lblContador.Text = "Clics: " & contador.ToString()`: El nuevo valor de `contador` se convierte a una cadena (usando `.ToString()`) y se concatena con "Clics: " para actualizar el texto de la etiqueta `lblContador`. Esto muestra al usuario el número actual de clics.

Programa 17. Formulario con dos botones para mostrar mensajes distintos

- **¿En qué consiste el ejercicio?** Este programa, similar al Programa 1, refuerza el concepto fundamental de cómo diferentes eventos de botón pueden desencadenar acciones distintas. En este caso, cada uno de los dos botones muestra un cuadro de mensaje (`MessageBox`) con un texto diferente.
- **Controles necesarios en el formulario y sus propiedades:**
    - **Button**:
        - **Name**: `btnMensaje1`.
            - **Propiedad**: Identificador único del primer botón.
            - **Utilización**: Al hacer clic, este botón muestra el "primer mensaje".
        - **Text**: "Mensaje 1".
            - **Propiedad**: Texto visible en el botón.
            - **Utilización**: Indica la acción asociada al botón.
    - **Button**:
        - **Name**: `btnMensaje2`.
            - **Propiedad**: Identificador único del segundo botón.
            - **Utilización**: Al hacer clic, este botón muestra el "segundo mensaje".
        - **Text**: "Mensaje 2".
            - **Propiedad**: Texto visible en el botón.
            - **Utilización**: Indica la acción asociada al botón.
- **Utilización desde el programa:**
    - Cada botón tiene su propio evento `_Click` asociado:
        - Cuando se hace clic en `btnMensaje1`, se ejecuta el evento `btnMensaje1_Click`. Dentro de él, `MessageBox.Show("Este es el primer mensaje")` despliega una ventana emergente con el texto especificado.
        - Cuando se hace clic en `btnMensaje2`, se ejecuta el evento `btnMensaje2_Click`. Dentro de él, `MessageBox.Show("Este es el segundo mensaje")` despliega otra ventana emergente con un texto diferente.
    - Esto ilustra cómo eventos distintos pueden llevar a acciones diferenciadas en la aplicación.

Programa 18. Formulario con un CheckBox que habilita o deshabilita un botón

- **¿En qué consiste el ejercicio?** Este programa ilustra una forma común de interacción entre controles: cómo un control (`CheckBox`) puede modificar el estado de otro control (`Button`), afectando su capacidad de ser utilizado (habilitado o deshabilitado).
- **Controles necesarios en el formulario y sus propiedades:**
    - **CheckBox**:
        - **Name**: `chkHabilitar`.
            - **Propiedad**: Identificador único del cuadro de verificación.

        - **Utilización**: Es el control que el usuario marca o desmarca para controlar el estado del botón de acción.
    - **Text**: "Habilitar botón".
        - **Propiedad**: Texto visible junto al `CheckBox` que describe su función.
    - **Checked**:
        - **Propiedad**: Una propiedad booleana que devuelve `True` si el `CheckBox` está marcado y `False` si no lo está.
        - **Utilización**: Su valor se usa directamente para establecer el estado `Enabled` del botón.
- **Button**:
    - **Name**: `btnAccion`.
        - **Propiedad**: Identificador único del botón de acción.
        - **Utilización**: Su estado de `Enabled` será controlado por el `CheckBox`.
    - **Text**: "Acción".
        - **Propiedad**: Texto visible del botón.
    - **Enabled**: `False` (inicialmente).
        - **Propiedad**: Una propiedad booleana que, si es `True`, permite que el control interactúe con el usuario (reciba clics, etc.). Si es `False`, el control aparece atenuado y no responde a las interacciones.
        - **Utilización**: Se establece su valor en función del estado del `CheckBox`.

- **Utilización desde el programa:**
    - El evento `chkHabilitar_CheckedChanged` se dispara cada vez que el estado de marcado del `CheckBox` `chkHabilitar` cambia (es decir, el usuario lo marca o desmarca).
    - Dentro de este evento, la línea `btnAccion.Enabled = chkHabilitar.Checked` realiza la lógica:
        - `chkHabilitar.Checked`: Obtiene el valor booleano actual del `CheckBox` (`True` si está marcado, `False` si no lo está).
        - Este valor booleano se asigna directamente a la propiedad `btnAccion.Enabled`.
    - Como resultado, el botón `btnAccion` se habilita automáticamente cuando el `CheckBox` está marcado y se deshabilita cuando el `CheckBox` está desmarcado.

Programa 19. Formulario con un TextBox multilinea y un botón para contar líneas

- **¿En qué consiste el ejercicio?** Este programa muestra cómo trabajar con un `TextBox` configurado para múltiples líneas y cómo obtener información estructurada de su contenido, específicamente el número total de líneas de texto ingresadas por el usuario.

- **Controles necesarios en el formulario y sus propiedades:**
    - **TextBox**:
        - **Name**: `txtMultilinea`.
            - **Propiedad**: Identificador único del cuadro de texto.
            - **Utilización**: Permite al usuario introducir texto en varias líneas, como en un editor de texto simple. Su contenido es el que se analizará.
        - **Multiline**: `True`.
            - **Propiedad**: Habilita al `TextBox` para aceptar y mostrar texto en múltiples líneas, en lugar de una sola.
            - **Utilización**: Fundamental para escenarios donde se espera una entrada de texto extensa.
        - **Size**: `200x100`.
            - **Propiedad**: Define las dimensiones (ancho x alto) del control en píxeles.
            - **Utilización**: Establece el tamaño visual del `TextBox` para que sea lo suficientemente grande para varias líneas.
    - **Button**:
        - **Name**: `btnContar`.
            - **Propiedad**: Identificador único del botón.
            - **Utilización**: Al hacer clic, desencadena la acción de contar las líneas en el `TextBox`.
        - **Text**: "Contar Líneas".
            - **Propiedad**: Texto visible del botón.
            - **Utilización**: Informa al usuario sobre la función del botón.
    - **Label**:
        - **Name**: `lblLineas`.
            - **Propiedad**: Identificador único de la etiqueta.
            - **Utilización**: Muestra el número de líneas calculado.
        - **Text**: "Líneas: 0".
            - **Propiedad**: Texto inicial de la etiqueta.
- **Utilización desde el programa:**
    - Al hacer clic en el botón `btnContar`, el evento `btnContar_Click` se activa.
    - Dentro de este evento, la línea `Dim lineas As Integer = txtMultilinea.Lines.Length` es clave:
        - `txtMultilinea.Lines`: Esta propiedad del `TextBox` devuelve un array de cadenas (`String()`), donde cada elemento del array corresponde a una línea de texto del `TextBox`.
        - `.Length`: Es una propiedad de los arrays que devuelve el número total de elementos que contiene el array. En este contexto, `txtMultilinea.Lines.Length` proporciona el número total de líneas en el `TextBox`.

- Finalmente, `lblLineas.Text = "Líneas: " & lineas.ToString()` convierte el número de líneas (`lineas`) a una cadena y lo muestra en la etiqueta `lblLineas`, junto con el prefijo "Líneas: ".

Programa 20. Formulario con una etiqueta que cambia de color al pasar el mouse

- **¿En qué consiste el ejercicio?** Este programa introduce los eventos relacionados con el mouse (`MouseEnter` y `MouseLeave`), permitiendo una interactividad visual dinámica. La aplicación cambia el color de la fuente de un `Label` cuando el puntero del mouse entra en su área y lo restaura cuando el puntero sale.
- **Controles necesarios en el formulario y sus propiedades:**
    - **Label**:
        - **Name**: `lblColor`.
            - **Propiedad**: Identificador único de la etiqueta.
            - **Utilización**: Muestra un texto. Su color de fuente (`ForeColor`) cambiará en respuesta a los eventos del mouse.
        - **Text**: "Pasa el mouse por aquí".
            - **Propiedad**: Texto visible de la etiqueta.
            - **Utilización**: Indica al usuario que debe interactuar con este control.
        - **ForeColor**:
            - **Propiedad**: Define el color de la fuente del texto mostrado en el control.
            - **Utilización**: Se modifica dinámicamente para dar un efecto visual de "hover" o interacción.
- **Utilización desde el programa:**
    - **lblColor_MouseEnter (Evento):** Este evento se dispara cuando el puntero del mouse del usuario **entra** en los límites visuales del control `lblColor`.
        - Dentro de este evento, `lblColor.ForeColor = Color.Red` cambia el color de la fuente de la etiqueta a rojo.
    - **lblColor_MouseLeave (Evento):** Este evento se dispara cuando el puntero del mouse del usuario **sale** de los límites visuales del control `lblColor`.
        - Dentro de este evento, `lblColor.ForeColor = Color.Black` restaura el color de la fuente de la etiqueta a negro (su color original o predeterminado).
    - `Color.Red` y `Color.Black` son constantes predefinidas de la clase `Color` para establecer colores específicos.

Programa 21. Formulario con un botón que cambia el ícono del formulario

- **¿En qué consiste el ejercicio?** Este programa demuestra cómo modificar el ícono de la ventana del formulario en tiempo de ejecución. El ícono es la pequeña imagen que aparece en la barra de título del formulario y en la barra de tareas de Windows.
- **Controles necesarios en el formulario y sus propiedades:**
    - **Button**:
        - **Name**: `btnIcono`.
            - **Propiedad**: Identificador único del botón.
            - **Utilización**: Al hacer clic, desencadena la acción de cambiar el ícono del formulario.
        - **Text**: "Cambiar Ícono".
            - **Propiedad**: Texto visible del botón.
            - **Utilización**: Informa al usuario sobre la función del botón.
    - **Formulario (Me)**:
        - **Icon**:
            - **Propiedad**: Representa el ícono asociado al formulario, que se muestra en la barra de título de la ventana y en la barra de tareas. Es un objeto de tipo `Icon`.
            - **Utilización**: Es la propiedad que se modifica para cambiar la representación visual del formulario en el sistema operativo.
- **Utilización desde el programa:**
    - Al hacer clic en el botón `btnIcono`, el evento `btnIcono_Click` se ejecuta.
    - La línea `Me.Icon = New Icon("icono.ico")` realiza el cambio:
        - `Me.Icon`: Hace referencia a la propiedad `Icon` de la instancia actual del formulario.
        - `New Icon("icono.ico")`: Crea una nueva instancia de la clase `Icon`, cargando la imagen desde el archivo "icono.ico".
            - **Requisito**: El archivo "icono.ico" debe existir en la misma carpeta que el ejecutable del programa o se debe especificar una ruta completa para que la carga sea exitosa.
    - Al asignar este nuevo objeto `Icon` a `Me.Icon`, el ícono visible del formulario en la barra de título y en la barra de tareas se actualiza.

Programa 22. Formulario con un TrackBar que controla el tamaño del texto

- **¿En qué consiste el ejercicio?** Este programa introduce el control `TrackBar` (barra deslizante) para permitir al usuario ajustar un valor de forma visual e interactiva. En este caso, el valor del `TrackBar` controla el tamaño de la fuente de un texto mostrado en un `Label`.
- **Controles necesarios en el formulario y sus propiedades:**
    - **Label**:
        - **Name**: `lblTexto`.

        - **Propiedad**: Identificador único de la etiqueta.
        - **Utilización**: Muestra un texto de ejemplo cuyo tamaño de fuente será modificado dinámicamente por el `TrackBar`.
    - **Text**: "Texto de ejemplo".
        - **Propiedad**: Contenido de texto inicial de la etiqueta.
    - **Font**:
        - **Propiedad**: Representa la fuente (tipo, estilo, tamaño) utilizada para mostrar el texto en el control. Es un objeto de tipo `Font`.
        - **Utilización**: Esta propiedad se recrea con un nuevo tamaño en función del valor del `TrackBar`.
- **TrackBar**:
    - **Name**: `trkTamaño`.
        - **Propiedad**: Identificador único del control `TrackBar`.
        - **Utilización**: Es el deslizador que el usuario mueve para seleccionar un valor numérico.
    - **Minimum**: `8`.
        - **Propiedad**: El valor más bajo que el `TrackBar` puede representar y que el deslizador puede alcanzar.
        - **Utilización**: Establece el tamaño mínimo de la fuente.
    - **Maximum**: `40`.
        - **Propiedad**: El valor más alto que el `TrackBar` puede representar y que el deslizador puede alcanzar.
        - **Utilización**: Establece el tamaño máximo de la fuente.
    - **Value**: `12` (inicial).
        - **Propiedad**: El valor entero actual seleccionado por el deslizador.
        - **Utilización**: Este valor se usa directamente como el tamaño de la fuente.
- **Utilización desde el programa:**
    - El evento `trkTamaño_Scroll` se dispara cada vez que el usuario mueve el deslizador del `TrackBar` `trkTamaño`.
    - Dentro de este evento, la línea `lblTexto.Font = New Font(lblTexto.Font.FontFamily, trkTamaño.Value)` actualiza el tamaño de la fuente:
        - `trkTamaño.Value`: Obtiene el valor numérico actual que el usuario ha seleccionado con el deslizador del `TrackBar`.
        - `New Font(lblTexto.Font.FontFamily, trkTamaño.Value)`: Crea un nuevo objeto `Font`. Mantiene la `FontFamily` (tipo de fuente, por ejemplo, "Arial") de la fuente actual de `lblTexto` (`lblTexto.Font.FontFamily`), pero usa el `trkTamaño.Value` como el nuevo tamaño de la fuente.
        - Este nuevo objeto `Font` se asigna a `lblTexto.Font`, haciendo que el texto de la etiqueta aumente o disminuya de tamaño visualmente a medida que el usuario desliza la barra.

Programa 23. Formulario con un cuadro de texto que valida correo electrónico

- **¿En qué consiste el ejercicio?** Este programa introduce el concepto de validación de datos del usuario utilizando expresiones regulares (`Regex`) para verificar si una cadena de texto ingresada en un `TextBox` tiene el formato típico de una dirección de correo electrónico válida.
- **Controles necesarios en el formulario y sus propiedades:**
    - **TextBox**:
        - **Name**: `txtCorreo`.
            - **Propiedad**: Identificador único del cuadro de texto.
            - **Utilización**: Permite al usuario ingresar la dirección de correo electrónico que se va a validar.
    - **Button**:
        - **Name**: `btnValidar`.
            - **Propiedad**: Identificador único del botón.
            - **Utilización**: Al hacer clic, desencadena el proceso de validación del correo electrónico.
        - **Text**: "Validar".
            - **Propiedad**: Texto visible del botón.
            - **Utilización**: Indica la acción principal del botón.
    - **Label**:
        - **Name**: `lblResultado`.
            - **Propiedad**: Identificador único de la etiqueta.
            - **Utilización**: Muestra el resultado de la validación, indicando si el correo es "válido" o "inválido".
        - **Text**: `""` (vacío inicialmente).
            - **Propiedad**: Contenido de texto inicial de la etiqueta.
- **Utilización desde el programa:**
    - **Imports System.Text.RegularExpressions**: Esta línea es necesaria al principio del archivo de código para poder utilizar las clases y métodos relacionados con las expresiones regulares, como la clase `Regex`.
    - Cuando se hace clic en el botón `btnValidar`, se ejecuta el evento `btnValidar_Click`.
    - Dentro de este evento:
        - `Dim patron As String = "^[^@\s]+@[^@\s]+\.[^@\s]+$"`: Se define una cadena que contiene la expresión regular (`patron`) utilizada para verificar el formato de un correo electrónico. Esta expresión asegura que haya un texto antes del '@', un texto entre '@' y '.', y un texto después del '.'.
        - `If Regex.IsMatch(txtCorreo.Text, patron) Then`:
            - **Regex.IsMatch(cadena, patron)**: Este es un método estático de la clase `Regex` que toma la cadena de texto a evaluar (`txtCorreo.Text`) y el patrón de expresión regular (`patron`). Devuelve `True` si la cadena coincide con el patrón, y `False` si no coincide.

        - **Utilización**: Es la función central para realizar la validación del formato.
    - Dependiendo del resultado de `Regex.IsMatch()`, la propiedad `lblResultado.Text` se actualiza para mostrar "Correo válido" o "Correo inválido", informando al usuario sobre el éxito o fracaso de la validación.

Programa 24. Formulario con un botón que rota imágenes

- **¿En qué consiste el ejercicio?** Este programa demuestra cómo manipular imágenes en una aplicación, específicamente rotarlas en un ángulo fijo, utilizando métodos proporcionados por la clase `Image`. La imagen se carga inicialmente y luego puede ser rotada al hacer clic en un botón.
- **Controles necesarios en el formulario y sus propiedades:**
    - **PictureBox**:
        - **Name**: `picImagen`.
            - **Propiedad**: Identificador único del control `PictureBox`.
            - **Utilización**: Es el contenedor donde se carga la imagen inicial y donde se mostrará la imagen después de ser rotada.
        - **SizeMode**: `StretchImage`.
            - **Propiedad**: Controla cómo la imagen se ajusta dentro del `PictureBox`. `StretchImage` redimensiona la imagen para que ocupe todo el espacio del control.
            - **Utilización**: Asegura que la imagen rotada se muestre visiblemente en el `PictureBox`, adaptándose a su tamaño.
        - **Size**: `200x200`.
            - **Propiedad**: Establece las dimensiones (ancho x alto) del `PictureBox` en píxeles.
            - **Utilización**: Define el área de visualización para la imagen.
    - **Button**:
        - **Name**: `btnRotar`.
            - **Propiedad**: Identificador único del botón.
            - **Utilización**: Al hacer clic, desencadena la rotación de la imagen.
        - **Text**: "Rotar".
            - **Propiedad**: Texto visible del botón.
            - **Utilización**: Indica la acción que el botón realizará.
- **Utilización desde el programa:**
    - **Dim imagen As Image = Image.FromFile("imagen.jpg")**: Se declara una variable de nivel de clase llamada `imagen` de tipo `Image`. En el momento de su declaración, se carga una imagen desde el archivo "imagen.jpg" (que debe estar disponible en la misma ruta que el ejecutable o tener una ruta completa especificada). Al ser de nivel de clase, esta variable `imagen` mantiene su estado entre las llamadas a los eventos, lo que es esencial para rotarla repetidamente.

- **Form1_Load (Evento):** Este evento se dispara cuando el formulario se carga inicialmente. Dentro de él, `picImagen.Image = imagen` asigna la imagen cargada a la propiedad `Image` del `PictureBox`, mostrando la imagen al iniciar la aplicación.
- Cuando se hace clic en el botón `btnRotar`, se ejecuta el evento `btnRotar_Click`.
- Dentro de este evento:
    - `imagen.RotateFlip(RotateFlipType.Rotate90FlipNone)`: Llama al método `RotateFlip()` de la variable `imagen`. Este método permite aplicar rotaciones y/o volteos a la imagen.
        - **RotateFlipType.Rotate90FlipNone**: Es una enumeración que especifica el tipo de rotación/volteo a aplicar. En este caso, indica una rotación de 90 grados en el sentido de las agujas del reloj, sin voltear la imagen.
        - **Utilización**: Modifica la imagen almacenada en la memoria.
    - `picImagen.Image = imagen`: Una vez que la imagen ha sido rotada en la memoria, se reasigna la misma variable `imagen` a la propiedad `Image` del `PictureBox`. Esto actualiza el `PictureBox` para mostrar la imagen en su nuevo estado rotado.

Programa 25. Formulario con una lista de opciones en un ComboBox

- **¿En qué consiste el ejercicio?** Este ejercicio introduce el control `ComboBox`, que proporciona una lista desplegable de opciones al usuario. El programa muestra cómo poblar esta lista con elementos predefinidos cuando el formulario se carga.
- **Controles necesarios en el formulario y sus propiedades:**
    - **ComboBox**:
        - **Name**: `cmbOpciones`.
            - **Propiedad**: Identificador único del control `ComboBox`.
            - **Utilización**: Es el control principal para mostrar una lista de opciones de las cuales el usuario puede seleccionar una.
        - **DropDownStyle**: `DropDownList`.
            - **Propiedad**: Controla el estilo del `ComboBox`. Cuando se establece en `DropDownList`, el usuario solo puede seleccionar uno de los elementos de la lista desplegable y no puede escribir su propio texto en la parte de entrada del `ComboBox`.
            - **Utilización**: Garantiza que el usuario elija únicamente entre las opciones proporcionadas, sin posibilidad de introducir valores arbitrarios.
        - **Items**:
            - **Propiedad**: Una colección que contiene todos los elementos que se muestran en la lista desplegable del `ComboBox`.

        - **Utilización**: Se usa para añadir programáticamente los elementos que el usuario podrá seleccionar.
- **Utilización desde el programa:**
    - El evento `Form1_Load` se dispara cuando el formulario se carga por primera vez y se hace visible. Es un lugar común para realizar la inicialización de controles.
    - Dentro de `Form1_Load`, las líneas `cmbOpciones.Items.Add("Opción 1")`, `cmbOpciones.Items.Add("Opción 2")`, `cmbOpciones.Items.Add("Opción 3")` añaden elementos a la colección `Items` del `cmbOpciones`.
        - **`Items.Add()`**: Este método añade una nueva cadena de texto como un elemento individual a la lista desplegable del `ComboBox`.
        - **Utilización**: Esto llena la lista desplegable con las opciones que el usuario puede ver y seleccionar, haciendo que el `ComboBox` esté listo para su interacción.

Programa 26. Formulario con un ListBox que agrega elementos desde un TextBox

- **¿En qué consiste el ejercicio?** Este programa muestra cómo el usuario puede añadir elementos a una lista dinámica visible en un control `ListBox`. Los elementos se ingresan a través de un `TextBox` y se agregan al `ListBox` al presionar un botón, limpiando el `TextBox` para una nueva entrada.
- **Controles necesarios en el formulario y sus propiedades:**
    - **TextBox**:
        - **Name**: `txtElemento`.
            - **Propiedad**: Identificador único del cuadro de texto.
            - **Utilización**: Permite al usuario escribir el texto del nuevo elemento que desea añadir a la lista. Su contenido es la fuente de datos para el `ListBox`.
        - **Text**:
            - **Propiedad**: Contiene la cadena de texto actual ingresada por el usuario.
            - **Utilización**: Se lee para obtener el elemento a añadir y se limpia después de la operación.
    - **Button**:
        - **Name**: `btnAgregar`.
            - **Propiedad**: Identificador único del botón.
            - **Utilización**: Desencadena la acción de agregar el texto del `TextBox` al `ListBox`.
        - **Text**: "Agregar".
            - **Propiedad**: Texto visible del botón.
            - **Utilización**: Indica la función principal del botón.
    - **ListBox**:

        - **Name**: `lstElementos`.
            - **Propiedad**: Identificador único del control `ListBox`.
            - **Utilización**: Muestra una lista de elementos al usuario. En este programa, nuevos elementos se añaden a esta lista dinámicamente.
        - **Items**:
            - **Propiedad**: Una colección que contiene todos los elementos que se muestran en el `ListBox`.
            - **Utilización**: Se usa para añadir los elementos ingresados por el usuario.
- **Utilización desde el programa:**
    - Al hacer clic en el botón `btnAgregar`, se ejecuta el evento `btnAgregar_Click`.
    - Dentro de este evento:
        - `If txtElemento.Text <> "" Then`: Se verifica si el `TextBox` `txtElemento` no está vacío. Esta es una buena práctica para evitar añadir elementos en blanco a la lista.
        - `lstElementos.Items.Add(txtElemento.Text)`: Si el `TextBox` contiene texto, este texto se añade como un nuevo elemento a la colección `Items` del `ListBox` `lstElementos`, haciéndolo visible en la lista.
        - `txtElemento.Clear()`: Después de añadir el elemento, el método `Clear()` se llama en `txtElemento` para vaciar su contenido. Esto prepara el `TextBox` para que el usuario pueda ingresar un nuevo elemento cómodamente.

Programa 27. Formulario con un botón que elimina el elemento seleccionado del ListBox

- **¿En qué consiste el ejercicio?** Este ejercicio complementa el programa anterior y demuestra cómo interactuar con los elementos de un `ListBox` para eliminarlos. Permite al usuario seleccionar un elemento de la lista y luego borrarlo al hacer clic en un botón.
- **Controles necesarios en el formulario y sus propiedades:**
    - **ListBox**:
        - **Name**: `lstElementos`.
            - **Propiedad**: Identificador único del control `ListBox`.
            - **Utilización**: Muestra una lista de elementos. El usuario puede seleccionar uno de estos elementos para su posterior eliminación.
        - **SelectedIndex**:
            - **Propiedad**: Un valor entero que indica el índice basado en cero del elemento seleccionado actualmente en el `ListBox`. Si no hay ningún elemento seleccionado, su valor es `-1`.
            - **Utilización**: Es crucial para determinar qué elemento ha sido elegido por el usuario para ser eliminado.

        - **Items**:
            - **Propiedad**: Una colección que contiene todos los elementos que se muestran en el `ListBox`.
            - **Utilización**: Se usa para eliminar un elemento de la lista en una posición específica.
    - **Button**:
        - **Name**: `btnEliminar`.
            - **Propiedad**: Identificador único del botón.
            - **Utilización**: Desencadena la acción de eliminar el elemento seleccionado del `ListBox`.
        - **Text**: "Eliminar Seleccionado".
            - **Propiedad**: Texto visible del botón.
            - **Utilización**: Indica claramente la función del botón.
- **Utilización desde el programa:**
    - Cuando se hace clic en el botón `btnEliminar`, se ejecuta el evento `btnEliminar_Click`.
    - Dentro de este evento:
        - `If lstElementos.SelectedIndex >= 0 Then`: Esta condición es fundamental para la robustez del programa. Verifica si hay algún elemento seleccionado en el `ListBox` (`lstElementos.SelectedIndex` devolverá un valor mayor o igual a 0 si hay una selección). Si no hay nada seleccionado, la operación de eliminación no se realiza, evitando un error.
        - `lstElementos.Items.RemoveAt(lstElementos.SelectedIndex)`: Si se ha seleccionado un elemento, este método se llama en la colección `Items` del `ListBox`.
            - **`RemoveAt()`**: Este método elimina el elemento de la colección en el índice especificado. Se le pasa el `lstElementos.SelectedIndex` para eliminar precisamente el elemento que el usuario ha seleccionado.
            - **Utilización**: Elimina visual y programáticamente el elemento seleccionado de la lista.

Programa 28. Formulario con un cuadro de texto que convierte a número romano

- **¿En qué consiste el ejercicio?** Este programa es un ejemplo de conversión de formatos numéricos más complejos. Aplica una lógica específica para transformar un número entero ingresado por el usuario a su representación en números romanos.
- **Controles necesarios en el formulario y sus propiedades:**
    - **TextBox**:
        - **Name**: `txtNumero`.
            - **Propiedad**: Identificador único del cuadro de texto.

        - **Utilización**: Permite al usuario introducir un número entero que se desea convertir a formato romano.
    - **Text**:
        - **Propiedad**: Contiene el número (como texto) ingresado por el usuario.
        - **Utilización**: Se lee para obtener el valor numérico a procesar.
- **Button**:
    - **Name**: `btnConvertir`.
        - **Propiedad**: Identificador único del botón.
        - **Utilización**: Desencadena el proceso de conversión de número entero a romano.
    - **Text**: "Convertir".
        - **Propiedad**: Texto visible del botón.
        - **Utilización**: Indica la acción del botón.
- **Label**:
    - **Name**: `lblRomano`.
        - **Propiedad**: Identificador único de la etiqueta.
        - **Utilización**: Muestra el resultado de la conversión a número romano, o un mensaje de error si la entrada no es válida.
    - **Text**: `""` (vacío inicialmente).
        - **Propiedad**: Contenido de texto inicial de la etiqueta.

- **Utilización desde el programa:**
    - **Función `ConvertirRomano(num As Integer) As String`**: Esta es una función personalizada definida en el formulario. Toma un número entero (`num`) como entrada y, utilizando dos arrays (`valores` con los valores numéricos estándar romanos y `simbolos` con sus representaciones de cadena), construye la representación romana mediante restas sucesivas del número hasta que este llega a cero.
    - Cuando se hace clic en el botón `btnConvertir`, el evento `btnConvertir_Click` se activa.
    - Dentro de este evento:
        - `Dim numero As Integer`: Se declara una variable entera para almacenar el número convertido.
        - `If Integer.TryParse(txtNumero.Text, numero) Then`:
            - **Integer.TryParse(cadena, variable_salida)**: Este método estático intenta convertir el texto de `txtNumero.Text` a un número entero. Si la conversión es exitosa, devuelve `True` y almacena el entero en la variable `numero`; si falla (ej., el texto no es un número), devuelve `False`.
            - **Utilización**: Es una forma segura de manejar la entrada de números desde un `TextBox`, evitando errores si el usuario ingresa texto no numérico.
        - Si la conversión es exitosa (`Integer.TryParse` devuelve `True`): `lblRomano.Text = ConvertirRomano(numero)` llama a la

función `ConvertirRomano` con el número ingresado y muestra el resultado de la conversión a números romanos en la etiqueta `lblRomano.Text`.
    - Si la conversión falla (`Integer.TryParse` devuelve `False`): `lblRomano.Text = "Número inválido"` muestra un mensaje de error indicando que la entrada no es un número válido.

Programa 29. Formulario con un botón que genera un número aleatorio

- **¿En qué consiste el ejercicio?** Este programa introduce el uso de números aleatorios para generar resultados impredecibles. Cada vez que el usuario presiona un botón, la aplicación genera y muestra un nuevo número entero aleatorio dentro de un rango específico.
- **Controles necesarios en el formulario y sus propiedades:**
    - **Button**:
        - **Name**: `btnAleatorio`.
            - **Propiedad**: Identificador único del botón.
            - **Utilización**: Es el control que el usuario presiona para generar un nuevo número aleatorio.
        - **Text**: "Generar".
            - **Propiedad**: Texto visible del botón.
            - **Utilización**: Indica la acción que el botón realizará.
    - **Label**:
        - **Name**: `lblNumero`.
            - **Propiedad**: Identificador único de la etiqueta.
            - **Utilización**: Muestra el número aleatorio generado por el programa.
        - **Text**: `""` (vacío inicialmente).
            - **Propiedad**: Contenido de texto inicial de la etiqueta.
- **Utilización desde el programa:**
    - **Dim rnd As New Random()**: Se declara una instancia de la clase `Random` con el nombre `rnd` a nivel de clase (fuera de cualquier evento). Esto es importante porque si se creara una nueva instancia de `Random` dentro del evento de clic cada vez, se podría obtener la misma secuencia de números si los clics son muy rápidos, ya que el generador se inicializaría con la misma semilla de tiempo. Al ser una instancia única, `rnd` genera una secuencia más variada.
    - Al hacer clic en el botón `btnAleatorio`, se ejecuta el evento `btnAleatorio_Click`.
    - Dentro de este evento, la línea `lblNumero.Text = "Número: " & rnd.Next(1, 101).ToString()` realiza la generación y visualización:
        - **`rnd.Next(1, 101)`**: Llama al método `Next()` de la instancia `rnd` de `Random`. Este método devuelve un número entero aleatorio.
            - El primer argumento (`1`) es el límite inferior (inclusivo).
            - El segundo argumento (`101`) es el límite superior (exclusivo).

        - **Utilización**: Esto significa que el número aleatorio generado estará en el rango de 1 a 100, ambos incluidos.
    - `.ToString()`: Convierte el número entero generado a una cadena de texto.
    - Esta cadena se concatena con "Número: " y se asigna a `lblNumero.Text`, actualizando el texto visible de la etiqueta con el nuevo número aleatorio.

Programa 30. Formulario con un temporizador que actualiza la hora

- **¿En qué consiste el ejercicio?** Este programa es idéntico al Programa 9, y sirve para reforzar la implementación del control `Timer` para actualizar información en intervalos regulares. En este caso, sigue mostrando la hora actual del sistema en una etiqueta, actualizándola cada segundo.
- **Controles necesarios en el formulario y sus propiedades:**
    - **Label**:
        - **Name**: `lblHora`.
            - **Propiedad**: Identificador único de la etiqueta.
            - **Utilización**: Se usa para mostrar la hora actual. Su texto se actualiza cada segundo.
        - **Text**: "Hora: ".
            - **Propiedad**: Texto inicial que muestra la etiqueta.
    - **Timer**:
        - **Name**: `Timer1`.
            - **Propiedad**: Identificador único del control `Timer`.
            - **Utilización**: Es el componente que genera eventos a intervalos regulares.
        - **Enabled**: `True`.
            - **Propiedad**: Una propiedad booleana que, cuando se establece en `True`, hace que el `Timer` comience a funcionar y dispare su evento `Tick`.
            - **Utilización**: Asegura que el `Timer` esté activo desde el inicio de la aplicación.
        - **Interval**: `1000` (milisegundos).
            - **Propiedad**: Define el tiempo en milisegundos que debe transcurrir entre cada disparo del evento `Tick` del `Timer`.
            - **Utilización**: Un valor de `1000` significa que el evento `Tick` se disparará cada 1 segundo, permitiendo una actualización constante de la hora.
- **Utilización desde el programa:**
    - El `Timer1` está habilitado (`Enabled: True`) y configurado para disparar su evento `Tick` cada 1000 milisegundos (1 segundo).
    - En cada ocurrencia del evento `Timer1_Tick`, la línea `lblHora.Text = "Hora: " & DateTime.Now.ToLongTimeString()` se ejecuta:

        - `DateTime.Now.ToLongTimeString()`: Obtiene la fecha y hora actuales del sistema (`DateTime.Now`) y las convierte a una cadena de texto que representa solo la hora en un formato largo.
        - Esta cadena de hora se concatena con "Hora: " y se asigna a `lblHora.Text`, actualizando el texto visible de la etiqueta cada segundo.

Programa 31. Formulario con un botón que oculta y muestra una etiqueta

- **Explicación del Ejercicio:** Este programa básico demuestra cómo **controlar la visibilidad de un control**, en este caso una etiqueta (`Label`), en un formulario de Visual Basic .NET mediante un simple botón. Al hacer clic en el botón, la etiqueta alternará entre estar visible y oculta.
- **Controles Necesarios:**
    - **Label:**
        - **Nombre (Name):** `lblTexto`. Este es el identificador único del control en el código, permitiendo acceder a sus propiedades y métodos.
        - **Texto (Text):** "Soy una etiqueta". Define el texto que se muestra inicialmente en la etiqueta, el cual será ocultado o mostrado.
        - **Propiedad `Visible`:** Esta es la propiedad fundamental en este ejercicio. Es de tipo booleano (`True`/`False`). Si se establece en `True`, la etiqueta es visible; si se establece en `False`, la etiqueta se oculta. El programa alterna el valor de esta propiedad (`lblTexto.Visible = Not lblTexto.Visible`) para cambiar su estado de visibilidad.
        - **Utilización:** La etiqueta se usa para contener y mostrar un mensaje de texto. En este programa, su principal función es servir como el objeto cuya visibilidad será manipulada.
    - **Button:**
        - **Nombre (Name):** `btnOcultarMostrar`. Identificador del botón en el código.
        - **Texto (Text):** "Ocultar/Mostrar". El texto que aparece en la superficie del botón, indicando su funcionalidad al usuario.
        - **Evento `Click`:** `btnOcultarMostrar_Click`. Este evento se dispara cuando el usuario hace clic en el botón. Contiene el código VB.NET que modifica la propiedad `Visible` de `lblTexto`.
        - **Utilización:** El botón es el elemento interactivo que el usuario pulsa para desencadenar la acción de alternar la visibilidad de la etiqueta.

Programa 32. Formulario con un botón que abre un nuevo formulario vacío

- **Explicación del Ejercicio:** Este programa introduce el concepto de **múltiples formularios** en una aplicación. Al hacer clic en un botón, se abre una nueva ventana (un formulario vacío) de la aplicación.
- **Controles Necesarios:**
    - **Button:**
        - **Nombre (Name):** `btnAbrir`. Identificador del botón.

        - **Texto (Text):** "Abrir Formulario". Texto visible en el botón que le indica al usuario su acción.
        - **Evento `Click`:** `btnAbrir_Click`. Este evento se activa cuando el usuario pulsa el botón. Dentro de este evento, se crean y se muestran nuevas instancias de formularios.
        - **Utilización:** El botón sirve como el disparador para la creación y apertura del nuevo formulario.
    - **Form (Implícito):**
        - Aunque no es un control que se arrastre directamente al formulario principal, se interactúa con una **nueva instancia de un `Form` genérico** creada en tiempo de ejecución (`Dim nuevoForm As New Form`).
        - **Propiedad `Text`:** "Nuevo Formulario". Se establece para darle un título a la barra de título de la nueva ventana que se abre.
        - **Método `Show()`:** Este método hace que la instancia del formulario sea visible en la pantalla.
        - **Utilización:** El nuevo formulario (`nuevoForm`) es la ventana secundaria que se presenta al usuario, demostrando cómo una aplicación puede tener múltiples interfaces.

Programa 33. Formulario con una barra de progreso que avanza al presionar un botón

- **Explicación del Ejercicio:** Este ejercicio utiliza el control **ProgressBar** para **visualizar el progreso de una tarea simulada**. Cada vez que el usuario presiona un botón, la barra de progreso avanza un cierto número de unidades.
- **Controles Necesarios:**
    - **ProgressBar:**
        - **Nombre (Name):** `prgBarra`. Identificador del control de barra de progreso.
        - **Propiedad `Minimum`:** 0. Define el valor mínimo del rango de la barra de progreso. El progreso comienza desde este valor.
        - **Propiedad `Maximum`:** 100. Define el valor máximo del rango de la barra de progreso. Cuando el `Value` alcanza este punto, la barra está completa.
        - **Propiedad `Value`:** 0. Representa el progreso actual de la barra. Este valor se incrementa programáticamente.
        - **Utilización:** La barra de progreso proporciona una retroalimentación visual al usuario sobre el avance de una operación, aunque en este caso es una simulación.
    - **Button:**
        - **Nombre (Name):** `btnAvanzar`. Identificador del botón.
        - **Texto (Text):** "Avanzar". Texto visible en el botón.
        - **Evento `Click`:** `btnAvanzar_Click`. Se dispara al hacer clic en el botón. El código dentro de este evento incrementa la propiedad `Value` de `prgBarra` en 10 unidades, siempre y cuando no exceda el `Maximum`.
        - **Utilización:** El botón es el elemento interactivo que el usuario pulsa para simular el avance de la tarea y, por ende, el progreso de la barra.

Programa 34. Formulario con un botón que copia texto al portapapeles

- **Explicación del Ejercicio:** Este programa demuestra cómo una aplicación puede **interactuar con el sistema operativo** para copiar texto a la memoria del portapapeles de Windows, haciéndolo disponible para ser pegado en otras aplicaciones.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtEntrada`. Identificador del cuadro de texto.
        - **Propiedad `Text`:** Contiene el texto que el usuario ingresa o que ya está presente en el `TextBox`. Este es el texto que se copiará al portapapeles.
        - **Utilización:** El `TextBox` es el origen del texto que se copiará al portapapeles.
    - **Button:**
        - **Nombre (Name):** `btnCopiar`. Identificador del botón.
        - **Texto (Text):** "Copiar". Texto visible en el botón.
        - **Evento `Click`:** `btnCopiar_Click`. Se activa al hacer clic en el botón. Contiene la instrucción `Clipboard.SetText(txtEntrada.Text)` que toma el texto de `txtEntrada` y lo coloca en el portapapeles del sistema.
        - **Utilización:** El botón desencadena la acción de copiar el texto al portapapeles.

Programa 35. Formulario con un cuadro de texto que limita caracteres

- **Explicación del Ejercicio:** Este programa demuestra una forma sencilla y directa de **restringir la cantidad de texto** que un usuario puede ingresar en un `TextBox`, utilizando una propiedad preexistente del control.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtLimitado`. Identificador del cuadro de texto.
        - **Propiedad `MaxLength`:** 10. Esta es la propiedad clave del ejercicio. Al establecer `MaxLength` a un valor específico (en este caso, 10) directamente en el diseñador de Visual Studio, el `TextBox` **automáticamente impide** que el usuario ingrese más caracteres de los permitidos, sin necesidad de escribir código VB.NET adicional para la validación de entrada.
        - **Propiedad `Text`:** Almacena el texto ingresado por el usuario, limitado por `MaxLength`.
        - **Utilización:** El `TextBox` es el campo de entrada donde el usuario escribe texto, con la restricción de longitud aplicada automáticamente por la propiedad `MaxLength`.
    - **Label:**
        - **Nombre (Name):** `lblAviso`. Identificador de la etiqueta.
        - **Texto (Text):** "Máximo 10 caracteres". Este texto sirve como una indicación visual para el usuario, informándole sobre la limitación del `TextBox`.
        - **Utilización:** La etiqueta es un elemento informativo pasivo que ayuda a guiar al usuario sobre el uso del `TextBox`.

Programa 36. Formulario con un botón que abre un cuadro de diálogo de color

- **Explicación del Ejercicio:** Este programa utiliza el control **ColorDialog** para permitir al usuario **seleccionar un color** de una paleta estándar de Windows y luego aplicar ese color al texto de un control `Label` en el formulario.
- **Controles Necesarios:**
    - **Button:**
        - **Nombre (Name):** `btnColor`. Identificador del botón.
        - **Texto (Text):** "Elegir Color". Texto visible en el botón.
        - **Evento `Click`:** `btnColor_Click`. Se dispara al hacer clic en el botón. Este evento es el que abre el `ColorDialog` y procesa la selección del usuario.
        - **Utilización:** El botón sirve como el disparador para abrir el cuadro de diálogo de selección de color.
    - **Label:**
        - **Nombre (Name):** `lblTexto`. Identificador de la etiqueta.
        - **Texto (Text):** "Texto de ejemplo". El texto que se mostrará en la etiqueta, y cuyo color de fuente (`ForeColor`) cambiará según la selección del usuario.
        - **Propiedad `ForeColor`:** Esta propiedad define el color del texto de la etiqueta. Se modificará con el color seleccionado por el usuario del `ColorDialog` (`ColorDialog1.Color`).
        - **Utilización:** La etiqueta es el control visual cuyo color de texto se personaliza en este programa.
    - **ColorDialog:**
        - **Nombre (Name):** `ColorDialog1`. Identificador del control de diálogo de color.
        - **Método `ShowDialog()`:** Este método abre la ventana estándar de selección de color de Windows. Retorna un `DialogResult` (como `DialogResult.OK` si el usuario hace clic en "Aceptar").
        - **Propiedad `Color`:** Después de que el usuario selecciona un color y hace clic en "Aceptar" en el cuadro de diálogo, esta propiedad contiene el valor del color RGB elegido. Este color se utiliza para actualizar la propiedad `ForeColor` del `Label`.
        - **Utilización:** El `ColorDialog` es el componente que facilita la interacción del usuario para elegir un color del sistema de forma gráfica.

Programa 37. Formulario con un botón que abre un cuadro de diálogo de fuente

- **Explicación del Ejercicio:** Similar al `ColorDialog`, este programa utiliza el control **FontDialog** para permitir al usuario **elegir una fuente** (tipo de letra, estilo y tamaño) de una lista estándar de Windows y aplicarla al texto de una etiqueta.
- **Controles Necesarios:**
    - **Button:**
        - **Nombre (Name):** `btnFuente`. Identificador del botón.
        - **Texto (Text):** "Elegir Fuente". Texto visible en el botón.

        - **Evento `Click`:** `btnFuente_Click`. Se activa al presionar el botón. Este evento abre el `FontDialog` y aplica la fuente seleccionada.
        - **Utilización:** El botón inicia el proceso de selección de fuente.
    - **Label:**
        - **Nombre (Name):** `lblTexto`. Identificador de la etiqueta.
        - **Texto (Text):** "Texto de ejemplo". El texto que se muestra en la etiqueta, cuya fuente cambiará.
        - **Propiedad `Font`:** Esta propiedad define el tipo de letra, estilo y tamaño del texto de la etiqueta. Se modificará con la fuente seleccionada por el usuario del `FontDialog` (`FontDialog1.Font`).
        - **Utilización:** La etiqueta es el control visual cuya fuente se personaliza.
    - **FontDialog:**
        - **Nombre (Name):** `FontDialog1`. Identificador del control de diálogo de fuente.
        - **Método `ShowDialog()`:** Este método abre el cuadro de diálogo estándar de fuentes de Windows. Retorna un `DialogResult`.
        - **Propiedad `Font`:** Después de que el usuario selecciona una fuente y hace clic en "Aceptar", esta propiedad contiene el objeto `Font` elegido, que se asigna a la propiedad `Font` del `Label`.
        - **Utilización:** El `FontDialog` es el componente que permite al usuario seleccionar una fuente del sistema de forma gráfica.

Programa 38. Formulario con un botón que abre un cuadro de diálogo de archivo

- **Explicación del Ejercicio:** Este programa permite al usuario **seleccionar un archivo** del sistema de archivos local utilizando el control **OpenFileDialog**, que es un cuadro de diálogo estándar de Windows. Una vez seleccionado, la ruta completa del archivo se muestra en un `MessageBox`.
- **Controles Necesarios:**
    - **Button:**
        - **Nombre (Name):** `btnArchivo`. Identificador del botón.
        - **Texto (Text):** "Abrir Archivo". Texto visible en el botón.
        - **Evento `Click`:** `btnArchivo_Click`. Se ejecuta cuando el usuario hace clic en el botón. Este evento es el que abre el `OpenFileDialog` y procesa el archivo seleccionado.
        - **Utilización:** El botón es el disparador para abrir el cuadro de diálogo de selección de archivo.
    - **OpenFileDialog:**
        - **Nombre (Name):** `OpenFileDialog1`. Identificador del control de diálogo de apertura de archivo.
        - **Método `ShowDialog()`:** Este método abre el cuadro de diálogo estándar de Windows para seleccionar un archivo. Retorna `DialogResult.OK` si el usuario selecciona un archivo y pulsa "Abrir".

        - **Propiedad `FileName`:** Después de que el usuario selecciona un archivo y confirma la acción (haciendo clic en "Abrir"), esta propiedad contiene la ruta completa del archivo seleccionado, incluyendo su nombre y extensión.
        - **Utilización:** El `OpenFileDialog` es el componente clave para permitir al usuario navegar y seleccionar un archivo en su sistema de archivos.

Programa 39. Formulario con un botón que abre un cuadro de diálogo de carpeta

- **Explicación del Ejercicio:** Este programa utiliza el control **FolderBrowserDialog** para permitir al usuario **seleccionar una carpeta** (directorio) del sistema de archivos. Después de la selección, la ruta de la carpeta elegida se muestra en un `MessageBox`.
- **Controles Necesarios:**
    - **Button:**
        - **Nombre (Name):** `btnCarpeta`. Identificador del botón.
        - **Texto (Text):** "Elegir Carpeta". Texto visible en el botón.
        - **Evento `Click`:** `btnCarpeta_Click`. Se activa cuando el usuario hace clic en el botón. Este evento abre el `FolderBrowserDialog` y procesa la carpeta seleccionada.
        - **Utilización:** El botón sirve como el disparador para abrir el cuadro de diálogo de selección de carpeta.
    - **FolderBrowserDialog:**
        - **Nombre (Name):** `FolderBrowserDialog1`. Identificador del control de diálogo de búsqueda de carpeta.
        - **Método `ShowDialog()`:** Este método abre el cuadro de diálogo estándar de Windows que permite al usuario navegar y seleccionar una carpeta. Retorna `DialogResult.OK` si el usuario selecciona una carpeta y hace clic en "Aceptar".
        - **Propiedad `SelectedPath`:** Después de que el usuario selecciona una carpeta y confirma la acción, esta propiedad contiene la ruta completa de la carpeta elegida.
        - **Utilización:** El `FolderBrowserDialog` es el componente esencial para permitir al usuario seleccionar una ubicación de carpeta en su sistema de archivos.

Programa 40. Formulario con un menú simple con dos opciones

- **Explicación del Ejercicio:** Este programa introduce el control **MenuStrip** para crear una **barra de menú** en la parte superior del formulario, similar a las aplicaciones de escritorio tradicionales. El menú contiene dos opciones simples que, al ser seleccionadas, muestran un mensaje diferente en un `MessageBox`.
- **Controles Necesarios:**
    - **MenuStrip:**
        - **Nombre (Name):** `MenuStrip1`. Este control se coloca en la parte superior del formulario y actúa como el contenedor principal para todos los elementos del menú.
        - **Utilización:** Proporciona una estructura de menú extensible para la aplicación, organizando las opciones de forma jerárquica.

- **ToolStripMenuItem (Opción 1):**
    - **Nombre (Name):** `opcion1ToolStripMenuItem`. Identificador para el primer elemento de menú. Es un subelemento que se anida bajo un menú principal (por ejemplo, "Archivo").
    - **Texto (Text):** "Opción 1". El texto que aparece en el menú para esta opción.
    - **Evento `Click`:** `opcion1ToolStripMenuItem_Click`. Se activa cuando el usuario hace clic en "Opción 1" del menú. El código dentro de este evento muestra un `MessageBox` específico para esta opción.
    - **Utilización:** Representa una opción seleccionable dentro del menú principal que ejecuta una acción específica.
- **ToolStripMenuItem (Opción 2):**
    - **Nombre (Name):** `opcion2ToolStripMenuItem`. Identificador para el segundo elemento de menú, anidado bajo el mismo menú principal.
    - **Texto (Text):** "Opción 2". El texto que aparece en el menú para esta opción.
    - **Evento `Click`:** `opcion2ToolStripMenuItem_Click`. Se activa cuando el usuario hace clic en "Opción 2". Muestra un `MessageBox` diferente para indicar que esta opción fue elegida.
    - **Utilización:** Similar a "Opción 1", es una acción seleccionable en el menú.

Programa 41. Formulario con un menú que cambia el color de fondo

- **Explicación del Ejercicio:** Este programa expande el uso del control `MenuStrip` para permitir al usuario **cambiar el color de fondo del formulario** de la aplicación seleccionando diferentes opciones de color desde un menú.
- **Controles Necesarios:**
    - **MenuStrip:**
        - **Nombre (Name):** `MenuStrip1`. El control que aloja los elementos del menú.
        - **Utilización:** Organiza las opciones de cambio de color en un formato de menú desplegable.
    - **ToolStripMenuItem (Rojo):**
        - **Nombre (Name):** `rojoToolStripMenuItem`. Identificador para la opción de menú "Rojo".
        - **Texto (Text):** "Rojo". Texto visible de la opción de menú.
        - **Evento `Click`:** `rojoToolStripMenuItem_Click`. Al seleccionar esta opción, la propiedad `BackColor` del formulario (`Me.BackColor`) se establece en `Color.Red`, cambiando el fondo a rojo.
        - **Utilización:** Permite al usuario aplicar un color rojo al fondo del formulario.
    - **ToolStripMenuItem (Verde):**
        - **Nombre (Name):** `verdeToolStripMenuItem`. Identificador para la opción de menú "Verde".
        - **Texto (Text):** "Verde". Texto visible de la opción de menú.
        - **Evento `Click`:** `verdeToolStripMenuItem_Click`. Al seleccionar esta opción, el `BackColor` del formulario se establece en `Color.Green`.
        - **Utilización:** Permite al usuario aplicar un color verde al fondo del formulario.
    - **ToolStripMenuItem (Azul):**

        - **Nombre (Name):** `azulToolStripMenuItem`. Identificador para la opción de menú "Azul".
        - **Texto (Text):** "Azul". Texto visible de la opción de menú.
        - **Evento `Click`:** `azulToolStripMenuItem_Click`. Al seleccionar esta opción, el `BackColor` del formulario se establece en `Color.Blue`.
        - **Utilización:** Permite al usuario aplicar un color azul al fondo del formulario.
    - **Form (Implícito):**
        - **Propiedad `BackColor`:** `Me.BackColor` se refiere al color de fondo del formulario actual. Esta propiedad es modificada por la selección del usuario en el menú.
        - **Utilización:** El formulario es el objeto cuya apariencia visual (color de fondo) se controla mediante las opciones del menú.

Programa 42. Formulario con un menú que muestra un mensaje

- **Explicación del Ejercicio:** Este es un ejemplo simple que demuestra cómo una opción dentro de un `MenuStrip` puede **desencadenar una acción básica**, como mostrar un cuadro de mensaje (`MessageBox`) al usuario.
- **Controles Necesarios:**
    - **MenuStrip:**
        - **Nombre (Name):** `MenuStrip1`. El control principal que contiene la estructura del menú.
        - **Utilización:** Aloja las opciones de menú para la aplicación.
    - **ToolStripMenuItem:**
        - **Nombre (Name):** `mensajeToolStripMenuItem`. Identificador del elemento de menú que, al ser clicado, mostrará un mensaje.
        - **Texto (Text):** "Mostrar Mensaje". El texto que el usuario ve en el menú para esta opción.
        - **Evento `Click`:** `mensajeToolStripMenuItem_Click`. Se ejecuta cuando el usuario selecciona esta opción del menú. El código dentro de este evento invoca `MessageBox.Show()` para desplegar el mensaje.
        - **Utilización:** Representa una opción del menú que proporciona información o una confirmación simple al usuario.

Programa 43. Formulario con un menú para abrir otro formulario

- **Explicación del Ejercicio:** Este programa integra la funcionalidad de **abrir un nuevo formulario** (una nueva ventana de la aplicación) a través de una opción de menú en un `MenuStrip`. Es similar al Programa 32, pero el disparador es un elemento de menú en lugar de un botón.
- **Controles Necesarios:**
    - **MenuStrip:**
        - **Nombre (Name):** `MenuStrip1`. El control que contiene la estructura del menú.
        - **Utilización:** Permite organizar las acciones de la aplicación, incluida la apertura de nuevas ventanas.
    - **ToolStripMenuItem:**

        - **Nombre (Name):** `abrirFormToolStripMenuItem`. Identificador de la opción de menú que abrirá el nuevo formulario.
        - **Texto (Text):** "Abrir Formulario". El texto que el usuario ve en el menú para esta opción.
        - **Evento `Click`:** `abrirFormToolStripMenuItem_Click`. Cuando se selecciona esta opción, se ejecuta el código que crea una nueva instancia de un formulario y lo hace visible.
        - **Utilización:** Proporciona un método accesible desde el menú para expandir la interfaz de usuario con nuevas ventanas.
    - **Form (Implícito):**
        - Se crea una nueva instancia de un **`Form` genérico** (`Dim nuevoForm As New Form`).
        - **Propiedad `Text`:** "Nuevo Formulario". Se asigna como el título de la barra de título de la ventana recién creada.
        - **Método `Show()`:** Este método hace que el nuevo formulario sea visible en la pantalla como una ventana independiente.
        - **Utilización:** El nuevo formulario es la ventana que se abre para el usuario, sirviendo como una interfaz adicional de la aplicación.

Programa 44. Formulario con una barra de herramientas (ToolStrip)

- **Explicación del Ejercicio:** Este programa introduce el control **ToolStrip**, que se utiliza para crear una **barra de herramientas** en el formulario. Esta barra contiene botones (ToolStripButton) que, al ser pulsados, realizan acciones específicas, ofreciendo una accesibilidad visual más directa que los menús.
- **Controles Necesarios:**
    - **ToolStrip:**
        - **Nombre (Name):** `ToolStrip1`. Este control se utiliza para agrupar y mostrar elementos como botones, cuadros de texto, etc., en una barra horizontal o vertical.
        - **Utilización:** Proporciona un conjunto de comandos y opciones de uso frecuente al alcance del usuario, a menudo con iconos para una rápida identificación.
    - **ToolStripButton (Nuevo):**
        - **Nombre (Name):** `btnNuevo`. Identificador del botón "Nuevo" en la barra de herramientas.
        - **Texto (Text):** "Nuevo". El texto que acompaña al botón, informando su función.
        - **Utilización:** Simula una acción para crear un nuevo elemento o iniciar una nueva tarea.
        - **Evento `Click`:** `btnNuevo_Click`. Se activa al hacer clic en este botón. En este ejemplo, muestra un `MessageBox` simple.
    - **ToolStripButton (Salir):**
        - **Nombre (Name):** `btnSalir`. Identificador del botón "Salir" en la barra de herramientas.
        - **Texto (Text):** "Salir". El texto que acompaña al botón.

- **Utilización:** Permite al usuario cerrar la aplicación de manera rápida.
- **Evento `Click`:** `btnSalir_Click`. Se activa al hacer clic. Utiliza `Application.Exit()` para terminar la ejecución de la aplicación, de forma similar al Programa 8.

Programa 45. Formulario con un cuadro de texto que busca en un ListBox

- **Explicación del Ejercicio:** Este programa permite al usuario **buscar un elemento específico** dentro de un control **ListBox** utilizando el texto que ingresa en un `TextBox`. Si el elemento es encontrado, se selecciona en la lista; si no, se notifica al usuario.
- **Controles Necesarios:**
  - **TextBox:**
    - **Nombre (Name):** `txtBuscar`. Identificador del cuadro de texto donde el usuario escribe el término de búsqueda.
    - **Propiedad `Text`:** Contiene la cadena de texto que el usuario desea encontrar en el `ListBox`.
    - **Utilización:** Es la interfaz de entrada para la consulta de búsqueda del usuario.
  - **Button:**
    - **Nombre (Name):** `btnBuscar`. Identificador del botón que inicia la búsqueda.
    - **Texto (Text):** "Buscar". Texto visible en el botón.
    - **Evento `Click`:** `btnBuscar_Click`. Se activa al hacer clic. Recupera el texto de `txtBuscar` y lo utiliza para buscar en `lstElementos`.
    - **Utilización:** Dispara la acción de buscar el texto en la lista.
  - **ListBox:**
    - **Nombre (Name):** `lstElementos`. Identificador del control de lista que contiene los elementos donde se realizará la búsqueda.
    - **Colección `Items`:** Contiene los elementos (cadenas de texto) que se muestran en la lista. Se recomienda poblar esta lista con algunos elementos previamente para probar la funcionalidad.
    - **Método `FindString(item)`:** Este método es esencial para la búsqueda. Busca la primera aparición del texto `item` (el contenido de `txtBuscar.Text`) dentro de la colección `Items` del `ListBox` y devuelve el índice de la coincidencia. Si no encuentra el texto, devuelve -1.
    - **Propiedad `SelectedIndex`:** Si `FindString` devuelve un índice válido (no -1), esta propiedad se establece a ese índice para que el elemento encontrado se muestre seleccionado visualmente en la lista.
    - **Utilización:** El `ListBox` es el repositorio de datos en el que se realiza la búsqueda y donde se muestra la selección resultante.

Programa 46. Formulario con un botón que cambia el título del formulario

- **Explicación del Ejercicio:** Este programa es un ejemplo básico de cómo modificar una propiedad fundamental del propio formulario en tiempo de ejecución: la propiedad `Text`, que controla el **texto que aparece en la barra de título de la ventana** de la aplicación.

- **Controles Necesarios:**
    - **Button:**
        - **Nombre (Name):** `btnTitulo`. Identificador del botón.
        - **Texto (Text):** "Cambiar Título". Texto visible en el botón.
        - **Evento `Click`:** `btnTitulo_Click`. Se dispara al hacer clic en el botón. Contiene el código para modificar la propiedad `Text` del formulario.
        - **Utilización:** El botón sirve como el disparador para cambiar el título del formulario.
    - **Form (Implícito):**
        - **Propiedad `Text`:** Se refiere al título del formulario actual. En el código, se accede a ella a través de `Me.Text`. Esta propiedad se actualiza a "Nuevo Título del Formulario" cuando se pulsa el botón.
        - **Utilización:** El formulario en sí es el objeto cuya propiedad `Text` se manipula, afectando la apariencia de la ventana de la aplicación.

Programa 47. Formulario con dos botones que mueven un Label en el formulario

- **Explicación del Ejercicio:** Este programa demuestra cómo **manipular la posición de un control** (`Label`) en el formulario en tiempo de ejecución, utilizando las propiedades que definen su ubicación. Con dos botones, el usuario puede mover la etiqueta hacia la izquierda o hacia la derecha.
- **Controles Necesarios:**
    - **Label:**
        - **Nombre (Name):** `lblMover`. Identificador de la etiqueta.
        - **Texto (Text):** "Mueve este texto". Texto visible en la etiqueta.
        - **Propiedad `Left`:** Esta propiedad representa la coordenada X (posición horizontal) del lado izquierdo del control `Label` en relación con su contenedor (el formulario). Al modificar este valor, se cambia la posición horizontal de la etiqueta. Un decremento la mueve a la izquierda, y un incremento la mueve a la derecha.
        - **Utilización:** La etiqueta es el elemento visual que el usuario mueve dentro del formulario.
    - **Button (Izquierda):**
        - **Nombre (Name):** `btnIzquierda`. Identificador del botón para mover a la izquierda.
        - **Texto (Text):** "Izquierda". Texto visible en el botón.
        - **Evento `Click`:** `btnIzquierda_Click`. Se activa al hacer clic en el botón. Decrementa la propiedad `Left` de `lblMover` en 10 píxeles (`lblMover.Left -= 10`), moviendo la etiqueta hacia la izquierda.
        - **Utilización:** Controla el movimiento de la etiqueta hacia la izquierda.
    - **Button (Derecha):**
        - **Nombre (Name):** `btnDerecha`. Identificador del botón para mover a la derecha.
        - **Texto (Text):** "Derecha". Texto visible en el botón.

        - **Evento `Click`:** `btnDerecha_Click`. Se activa al hacer clic en el botón. Incrementa la propiedad `Left` de `lblMover` en 10 píxeles (`lblMover.Left += 10`), moviendo la etiqueta hacia la derecha.
        - **Utilización:** Controla el movimiento de la etiqueta hacia la derecha.

Programa 48. Formulario con un botón que muestra la fecha actual

- **Explicación del Ejercicio:** Este programa utiliza la clase `DateTime` de .NET para obtener la **fecha actual del sistema** y mostrarla formateada en un control `Label` cuando el usuario hace clic en un botón.
- **Controles Necesarios:**
    - **Button:**
        - **Nombre (Name):** `btnFecha`. Identificador del botón.
        - **Texto (Text):** "Mostrar Fecha". Texto visible en el botón.
        - **Evento `Click`:** `btnFecha_Click`. Se ejecuta cuando el usuario presiona el botón. Contiene el código para obtener y mostrar la fecha actual.
        - **Utilización:** El botón sirve como el disparador para obtener y mostrar la fecha actual.
    - **Label:**
        - **Nombre (Name):** `lblFecha`. Identificador de la etiqueta.
        - **Propiedad `Text`:** Se actualiza para mostrar la fecha actual del sistema.
        - **Uso de `DateTime.Now`:** Esta propiedad estática de la clase `DateTime` devuelve un objeto `DateTime` que representa la fecha y hora actuales del sistema.
        - **Uso de `.ToLongDateString()`:** Este método convierte la parte de la fecha del objeto `DateTime` en una cadena de texto larga y legible, específica de la configuración regional del sistema (por ejemplo, "lunes, 1 de enero de 2024").
        - **Utilización:** La etiqueta es el control donde se presenta visualmente la fecha actual al usuario.

Programa 49. Formulario con un PictureBox que muestra diferentes imágenes

- **Explicación del Ejercicio:** Este programa demuestra cómo **cargar y visualizar imágenes** desde archivos locales en un control **PictureBox**. Mediante el uso de dos botones, el usuario puede seleccionar qué imagen se muestra en el `PictureBox`. Es una funcionalidad idéntica al Programa 11.
- **Controles Necesarios:**
    - **PictureBox:**
        - **Nombre (Name):** `picImagen`. Identificador del control que se usa para mostrar las imágenes.
        - **Propiedad `SizeMode`:** `StretchImage`. Esta propiedad determina cómo se ajusta la imagen dentro de los límites del `PictureBox`. `StretchImage` redimensiona la imagen para que llene todo el espacio del control, estirándola o encogiéndola si es necesario.

        - **Propiedad `Size`:** 200x200. Establece las dimensiones iniciales del `PictureBox` en píxeles.
        - **Propiedad `Image`:** Esta es la propiedad principal del `PictureBox`. Se le asigna un objeto `Image` (cargado desde un archivo) para que la imagen sea visible en el control.
        - **Utilización:** El `PictureBox` es el lienzo visual donde se cargan y exhiben las imágenes seleccionadas por el usuario.
    - **Button (Imagen 1):**
        - **Nombre (Name):** `btnImagen1`. Identificador del botón para cargar la primera imagen.
        - **Texto (Text):** "Imagen 1". Texto visible en el botón.
        - **Evento `Click`:** `btnImagen1_Click`. Al hacer clic, carga el archivo "imagen1.jpg" utilizando `Image.FromFile()` y lo asigna a `picImagen.Image`.
        - **Utilización:** Permite al usuario seleccionar y mostrar "imagen1.jpg".
    - **Button (Imagen 2):**
        - **Nombre (Name):** `btnImagen2`. Identificador del botón para cargar la segunda imagen.
        - **Texto (Text):** "Imagen 2". Texto visible en el botón.
        - **Evento `Click`:** `btnImagen2_Click`. Al hacer clic, carga el archivo "imagen2.jpg" y lo asigna a `picImagen.Image`.
        - **Uso de `Image.FromFile()`:** Este método estático de la clase `Image` es el encargado de cargar un archivo de imagen desde el disco. Se asume que los archivos "imagen1.jpg" y "imagen2.jpg" están ubicados en la misma carpeta que el ejecutable de la aplicación o se especifica una ruta completa.
        - **Utilización:** Permite al usuario seleccionar y mostrar "imagen2.jpg".

Programa 50. Formulario con un botón que guarda texto en un archivo

- **Explicación del Ejercicio:** Este programa introduce la funcionalidad de **guardar datos (texto)** ingresados por el usuario en un `TextBox` a un archivo en el disco duro. Para ello, utiliza el control **SaveFileDialog**, que permite al usuario elegir la ubicación y el nombre del archivo de destino.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtContenido`. Identificador del cuadro de texto.
        - **Propiedad `Multiline`:** `True`. Permite que el `TextBox` muestre y acepte múltiples líneas de texto, siendo adecuado para contenido extenso.
        - **Propiedad `Size`:** 200x100. Establece las dimensiones iniciales del `TextBox`.
        - **Propiedad `Text`:** Contiene el texto que el usuario ha ingresado y que se desea guardar en un archivo.
        - **Utilización:** El `TextBox` es el origen del texto que se va a guardar.
    - **Button:**
        - **Nombre (Name):** `btnGuardar`. Identificador del botón.

        - **Texto (Text):** "Guardar". Texto visible en el botón.
        - **Evento `Click`:** `btnGuardar_Click`. Se ejecuta al hacer clic en el botón. Este evento es el que abre el `SaveFileDialog` y, si el usuario confirma, guarda el contenido del `TextBox` en el archivo.
        - **Utilización:** El botón inicia la acción de guardar el contenido del `TextBox` en un archivo.
    - **SaveFileDialog:**
        - **Nombre (Name):** `SaveFileDialog1`. Identificador del control de diálogo para guardar archivos.
        - **Método `ShowDialog()`:** Abre el cuadro de diálogo estándar de Windows que permite al usuario navegar por las carpetas, especificar un nombre de archivo y elegir dónde guardar el archivo. Retorna `DialogResult.OK` si el usuario confirma el guardado.
        - **Propiedad `FileName`:** Después de que el usuario selecciona una ruta y nombre de archivo en el cuadro de diálogo, esta propiedad contiene la ruta completa del archivo donde se guardarán los datos.
        - **Uso de `System.IO.File.WriteAllText()`:** Para usar esta funcionalidad, se necesita importar el espacio de nombres `System.IO`. Este método estático escribe todo el contenido de una cadena de texto (el `txtContenido.Text`) en un archivo especificado por su ruta (`SaveFileDialog1.FileName`). Si el archivo no existe, lo crea; si existe, lo sobrescribe.
        - **Utilización:** El `SaveFileDialog` es crucial para permitir al usuario interactuar con el sistema de archivos para guardar el texto.

Programa 51. Agenda de contactos básica con ListBox

- **Explicación del Ejercicio:** Este programa representa la primera incursión en aplicaciones intermedias, creando una **agenda simple** que permite al usuario agregar nombres de contactos a un control **ListBox**. Incluye una verificación para asegurarse de que el `TextBox` de entrada no esté vacío.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtNombre`. Identificador del cuadro de texto donde el usuario ingresará el nombre del contacto.
        - **Propiedad `Text`:** Almacena el texto (nombre del contacto) que el usuario escribe. El programa verifica que esta propiedad no esté vacía antes de añadir el contacto.
        - **Método `Clear()`:** Después de que un nombre se añade a la lista, este método se invoca para vaciar el `TextBox`, preparándolo para la entrada del siguiente contacto.
        - **Utilización:** Es el campo de entrada para los nombres de los contactos que se agregarán a la agenda.
    - **Button:**

        - **Nombre (Name):** `btnAgregar`. Identificador del botón para añadir el contacto.
        - **Texto (Text):** "Agregar". Texto visible en la superficie del botón.
        - **Evento `Click`:** `btnAgregar_Click`. Se dispara cuando el usuario hace clic en el botón. Contiene la lógica para validar el `TextBox` y añadir el contenido al `ListBox`.
        - **Utilización:** El botón es el elemento interactivo que el usuario pulsa para registrar un nuevo contacto.
    - **ListBox:**
        - **Nombre (Name):** `lstContactos`. Identificador del control de lista que mostrará los nombres de los contactos agregados.
        - **Colección `Items`:** Esta colección almacena todos los elementos (cadenas de texto, en este caso, los nombres de los contactos) que se visualizan en el `ListBox`.
        - **Método `Items.Add(txtNombre.Text)`:** Añade el texto actual de `txtNombre` como un nuevo elemento al final de la colección `Items` del `ListBox`, haciendo que el nuevo contacto aparezca en la lista.
        - **Utilización:** El `ListBox` es el componente principal para mostrar la lista de contactos de la agenda al usuario.

Programa 52. Calculadora simple (suma, resta, multiplicación, división)

- **Explicación del Ejercicio:** Este programa implementa una **calculadora básica** capaz de realizar las cuatro operaciones aritméticas fundamentales (suma, resta, multiplicación y división). El usuario ingresa dos números en `TextBox` y luego selecciona la operación deseada mediante botones, mostrando el resultado en un `Label`.
- **Controles Necesarios:**
    - **TextBox (Número 1):**
        - **Nombre (Name):** `txtNum1`. Identificador del primer cuadro de texto para la entrada de números.
        - **Propiedad `Text`:** Contiene el primer número ingresado por el usuario. Su contenido se convierte a un valor numérico (`Double`) utilizando la función `Val()` antes de realizar cualquier operación.
        - **Utilización:** Es el campo de entrada para el primer operando de la operación.
    - **TextBox (Número 2):**
        - **Nombre (Name):** `txtNum2`. Identificador del segundo cuadro de texto para la entrada de números.
        - **Propiedad `Text`:** Contiene el segundo número ingresado por el usuario. También se convierte a `Double` con `Val()`. En el caso de la división, se incluye una validación crucial para verificar que su valor no sea cero (`Val(txtNum2.Text) <> 0`) para evitar errores de división por cero.
        - **Utilización:** Es el campo de entrada para el segundo operando.
    - **Button (Sumar):**
        - **Nombre (Name):** `btnSumar`. Identificador del botón de suma.
        - **Texto (Text):** "Sumar". Texto visible en el botón.

        - **Evento `Click`:** `btnSumar_Click`. Al hacer clic, calcula la suma de los valores de `txtNum1` y `txtNum2`.
        - **Utilización:** Desencadena la operación de suma.
    - **Button (Restar):**
        - **Nombre (Name):** `btnRestar`. Identificador del botón de resta.
        - **Texto (Text):** "Restar". Texto visible en el botón.
        - **Evento `Click`:** `btnRestar_Click`. Al hacer clic, calcula la resta.
        - **Utilización:** Desencadena la operación de resta.
    - **Button (Multiplicar):**
        - **Nombre (Name):** `btnMultiplicar`. Identificador del botón de multiplicación.
        - **Texto (Text):** "Multiplicar". Texto visible en el botón.
        - **Evento `Click`:** `btnMultiplicar_Click`. Al hacer clic, calcula la multiplicación.
        - **Utilización:** Desencadena la operación de multiplicación.
    - **Button (Dividir):**
        - **Nombre (Name):** `btnDividir`. Identificador del botón de división.
        - **Texto (Text):** "Dividir". Texto visible en el botón.
        - **Evento `Click`:** `btnDividir_Click`. Al hacer clic, calcula la división. Incluye una lógica `If` para comprobar si `txtNum2` es cero, mostrando un mensaje de error si lo es, y realizando la división si no.
        - **Utilización:** Desencadena la operación de división, con manejo de errores.
    - **Label:**
        - **Nombre (Name):** `lblResultado`. Identificador de la etiqueta.
        - **Propiedad `Text`:** Se utiliza para mostrar el resultado de la operación aritmética o cualquier mensaje de error (como "Error: división por 0").
        - **Uso de `Val()`:** Convierte el texto de los `TextBox` a valores numéricos (Double) para poder realizar las operaciones aritméticas.
        - **Uso de `.ToString()`:** Convierte el resultado numérico de la operación a una cadena de texto para que pueda ser asignado y mostrado en la propiedad `Text` del `Label`.
        - **Utilización:** Muestra al usuario el resultado de la calculadora.

Programa 53. Conversor de temperatura (Celsius-Fahrenheit)

- **Explicación del Ejercicio:** Este programa proporciona una utilidad para convertir una temperatura dada en **grados Celsius a su equivalente en grados Fahrenheit**. El usuario ingresa la temperatura en Celsius en un `TextBox`, y el resultado de la conversión se muestra en un `Label` al hacer clic en un botón.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtCelsius`. Identificador del cuadro de texto para la entrada de la temperatura en Celsius.

        - **Propiedad `Text`:** Contiene el valor numérico de la temperatura en Celsius ingresado por el usuario. Este valor se convierte a `Double` utilizando la función `Val()` para realizar el cálculo.
        - **Utilización:** Es el campo donde el usuario introduce la temperatura a convertir.
    - **Button:**
        - **Nombre (Name):** `btnConvertir`. Identificador del botón de conversión.
        - **Texto (Text):** "Convertir". Texto visible en el botón.
        - **Evento `Click`:** `btnConvertir_Click`. Se ejecuta al hacer clic. Dentro de este evento, se recupera el valor de `txtCelsius`, se aplica la fórmula de conversión y el resultado se muestra en `lblResultado`.
        - **Utilización:** Dispara la acción de conversión de temperatura.
    - **Label:**
        - **Nombre (Name):** `lblResultado`. Identificador de la etiqueta.
        - **Propiedad `Text`:** Se utiliza para mostrar la temperatura original en Celsius y su equivalente calculado en Fahrenheit, en un formato legible (ej. "25 °C = 77 °F").
        - **Uso de `Val()`:** Convierte el texto de `txtCelsius` a un valor numérico (`Double`) para que pueda ser utilizado en la fórmula de conversión.
        - **Utilización:** Presenta el resultado de la conversión al usuario.

Programa 54. Conversor de divisas

- **Explicación del Ejercicio:** Este programa implementa un conversor de divisas sencillo que transforma una cantidad de **Euros a Dólares Estadounidenses (USD)**. Utiliza una tasa de cambio fija predefinida. El usuario introduce la cantidad en Euros, y el programa muestra el equivalente en USD en un `Label`.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtEuros`. Identificador del cuadro de texto para la entrada de la cantidad en Euros.
        - **Propiedad `Text`:** Contiene el valor numérico de los Euros ingresados por el usuario. Se convierte a `Double` con `Val()` para el cálculo.
        - **Utilización:** Es el campo de entrada donde el usuario especifica la cantidad de Euros a convertir.
    - **Button:**
        - **Nombre (Name):** `btnConvertir`. Identificador del botón de conversión.
        - **Texto (Text):** "Convertir a USD". Texto visible en el botón.
        - **Evento `Click`:** `btnConvertir_Click`. Se activa al hacer clic. Este evento recupera el valor de `txtEuros`, aplica la tasa de cambio fija y muestra el resultado.
        - **Utilización:** Dispara la acción de convertir la divisa.
    - **Label:**
        - **Nombre (Name):** `lblResultado`. Identificador de la etiqueta.

        - **Propiedad `Text`:** Se utiliza para mostrar la cantidad original en Euros y su equivalente en Dólares, incluyendo el símbolo de la moneda, en un formato legible (ej. "100 € = 110 $").
        - **Uso de `Val()`:** Convierte el texto de `txtEuros` a un valor numérico (`Double`) para realizar el cálculo de conversión.
        - **Utilización:** Presenta el resultado de la conversión de divisas al usuario.

Programa 55. Bloc de notas básico

- **Explicación del Ejercicio:** Este programa es un **editor de texto extremadamente simple**, cuya funcionalidad se basa principalmente en las propiedades de un único control `TextBox`. El `TextBox` se configura para ser multilinea y ocupar todo el espacio disponible del formulario, creando una interfaz de bloc de notas básica sin necesidad de código VB.NET adicional.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtContenido`. Identificador del cuadro de texto principal.
        - **Propiedad `Multiline`:** `True`. Esta es una propiedad fundamental para un bloc de notas. Al establecerla en `True`, el `TextBox` puede mostrar y aceptar varias líneas de texto, permitiendo al usuario escribir contenido extenso y que el texto se ajuste automáticamente a nuevas líneas.
        - **Propiedad `Dock`:** `Fill`. Esta propiedad es clave para la interfaz. Cuando se establece en `Fill`, el `TextBox` se ajusta y expande automáticamente para ocupar todo el espacio de su contenedor (en este caso, el formulario completo), lo que proporciona una gran área de texto, ideal para un bloc de notas.
        - **Propiedad `Text`:** Almacena el texto que el usuario escribe en el `TextBox`.
        - **Utilización:** El `TextBox` es el único componente funcional visible para el usuario, sirviendo como el área principal para escribir y editar texto. No se requiere código para este programa, ya que las propiedades configuradas en el diseñador son suficientes para su propósito.

Programa 56. Aplicación que guarda y carga texto desde archivo

- **Explicación del Ejercicio:** Este programa amplía la funcionalidad de un bloc de notas, permitiendo al usuario **guardar el texto** ingresado en un `TextBox` en un archivo y **cargar texto** desde un archivo existente de nuevo al `TextBox`. Utiliza los controles `OpenFileDialog` y `SaveFileDialog` para la interacción con el sistema de archivos.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtContenido`. Identificador del cuadro de texto principal.
        - **Propiedad `Multiline`:** `True`. Permite que el `TextBox` maneje múltiples líneas de texto, adecuado para la edición de archivos de texto.

    - **Propiedad `Size`:** 300x200. Define las dimensiones del `TextBox` en el formulario.
    - **Propiedad `Text`:** Contiene el texto que se desea guardar o que se ha cargado desde un archivo.
    - **Utilización:** Es el área principal para la edición y visualización del texto.
- **Button (Guardar):**
    - **Nombre (Name):** `btnGuardar`. Identificador del botón para guardar el texto.
    - **Texto (Text):** "Guardar". Texto visible en el botón.
    - **Evento `Click`:** `btnGuardar_Click`. Se activa al hacer clic. Abre el `SaveFileDialog` y, si se confirma la selección, utiliza `File.WriteAllText()` para guardar el contenido de `txtContenido`.
    - **Utilización:** Permite al usuario guardar el contenido del `TextBox` en un archivo.
- **Button (Cargar):**
    - **Nombre (Name):** `btnCargar`. Identificador del botón para cargar texto.
    - **Texto (Text):** "Cargar". Texto visible en el botón.
    - **Evento `Click`:** `btnCargar_Click`. Se activa al hacer clic. Abre el `OpenFileDialog` y, si se selecciona un archivo, utiliza `File.ReadAllText()` para cargar su contenido en `txtContenido`.
    - **Utilización:** Permite al usuario cargar texto desde un archivo al `TextBox`.
- **OpenFileDialog:**
    - **Nombre (Name):** `OpenFileDialog1`. Identificador del control de diálogo para abrir archivos.
    - **Método `ShowDialog()`:** Abre el cuadro de diálogo estándar de Windows para seleccionar un archivo. Retorna `DialogResult.OK` si el usuario selecciona y abre un archivo.
    - **Propiedad `FileName`:** Contiene la ruta completa del archivo seleccionado por el usuario.
    - **Uso de `System.IO.File.ReadAllText(OpenFileDialog1.FileName)`:** Para usar esta función, se debe importar `System.IO`. Lee todo el contenido de un archivo de texto en la ruta especificada y lo devuelve como una cadena.
    - **Utilización:** Facilita la selección de un archivo existente para cargar su contenido.
- **SaveFileDialog:**
    - **Nombre (Name):** `SaveFileDialog1`. Identificador del control de diálogo para guardar archivos.
    - **Método `ShowDialog()`:** Abre el cuadro de diálogo estándar para guardar un archivo. Retorna `DialogResult.OK` si el usuario especifica una ubicación y nombre y confirma el guardado.
    - **Propiedad `FileName`:** Contiene la ruta completa del archivo donde se guardará.

        - **Uso de `System.IO.File.WriteAllText(SaveFileDialog1.FileName, txtContenido.Text)`:** Escribe la cadena de texto proporcionada (`txtContenido.Text`) en el archivo especificado.
        - **Utilización:** Permite al usuario elegir dónde y con qué nombre guardar el archivo.

Programa 57. Aplicación que cuenta palabras en un TextBox

- **Explicación del Ejercicio:** Este programa está diseñado para **contar el número de palabras** en un `TextBox` que puede contener múltiples líneas de texto. Al hacer clic en un botón, el programa procesa el contenido del `TextBox` y muestra el recuento de palabras en un `Label`.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtTexto`. Identificador del cuadro de texto donde el usuario escribe el contenido.
        - **Propiedad `Multiline`:** `True`. Permite la entrada y visualización de texto en varias líneas, lo que es común al trabajar con documentos o textos largos.
        - **Propiedad `Size`:** 300x100. Define las dimensiones del `TextBox` en el formulario.
        - **Propiedad `Text`:** Contiene el texto ingresado por el usuario. Este es el contenido que se analizará para contar las palabras.
        - **Uso de `Text.Split({" "c, vbCrLf}, StringSplitOptions.RemoveEmptyEntries)`:** Esta es una parte clave de la lógica. El método `Split()` de la cadena de texto divide el contenido del `TextBox` en un array de cadenas (palabras). Se usan múltiples delimitadores: el espacio (`" "c`) y el salto de línea (`vbCrLf`). `StringSplitOptions.RemoveEmptyEntries` asegura que los espacios múltiples o líneas vacías no se cuenten como palabras individuales, mejorando la precisión del conteo.
        - **Utilización:** Sirve como el área de entrada para el texto a analizar.
    - **Button:**
        - **Nombre (Name):** `btnContar`. Identificador del botón que inicia la acción de contar.
        - **Texto (Text):** "Contar Palabras". Texto visible en el botón.
        - **Evento `Click`:** `btnContar_Click`. Se ejecuta cuando el usuario hace clic. Llama a la lógica para dividir el texto y contar las palabras.
        - **Utilización:** Dispara el proceso de conteo de palabras.
    - **Label:**
        - **Nombre (Name):** `lblResultado`. Identificador de la etiqueta.
        - **Propiedad `Text`:** Se utiliza para mostrar el número final de palabras encontradas en el `TextBox` (ej. "Número de palabras: 50").
        - **Uso de `palabras.Length`:** Una vez que el texto se ha dividido en el array `palabras`, la propiedad `Length` de este array devuelve la cantidad de

elementos (es decir, el número de palabras), que luego se convierte a cadena para mostrarse.
- **Utilización:** Presenta el resultado del conteo al usuario.

Programa 58. Aplicación que muestra tablas de multiplicar

- **Explicación del Ejercicio:** Este programa permite al usuario ingresar un número en un `TextBox` y, al hacer clic en un botón, **genera y muestra la tabla de multiplicar** de ese número (del 1 al 10) en un control `ListBox`. La lista se limpia antes de cada nueva generación para mostrar solo la tabla actual.
- **Controles Necesarios:**
  - **TextBox:**
    - **Nombre (Name):** `txtNumero`. Identificador del cuadro de texto donde el usuario ingresará el número para la tabla de multiplicar.
    - **Propiedad `Text`:** Contiene el valor numérico ingresado por el usuario. Se convierte a un entero (`Integer`) con la función `Val()` para realizar los cálculos.
    - **Utilización:** Es el campo de entrada para el número base de la tabla.
  - **Button:**
    - **Nombre (Name):** `btnGenerar`. Identificador del botón que inicia la generación de la tabla.
    - **Texto (Text):** "Generar Tabla". Texto visible en el botón.
    - **Evento `Click`:** `btnGenerar_Click`. Se activa al hacer clic. Contiene la lógica para limpiar la lista, obtener el número del `TextBox`, y luego usar un bucle `For` para calcular y añadir cada línea de la tabla de multiplicar al `ListBox`.
    - **Utilización:** Dispara el proceso de generación de la tabla de multiplicar.
  - **ListBox:**
    - **Nombre (Name):** `lstTabla`. Identificador del control de lista que mostrará la tabla de multiplicar generada.
    - **Colección `Items`:** Esta colección almacena las cadenas de texto que representan cada línea de la tabla (ej. "5 x 1 = 5").
    - **Método `Items.Clear()`:** Se llama al inicio del evento `btnGenerar_Click` para eliminar cualquier tabla de multiplicar anterior que pudiera estar mostrándose, asegurando que solo la tabla actual sea visible.
    - **Método `Items.Add(n & " x " & i & " = " & (n * i))`:** Dentro del bucle `For`, esta instrucción construye una cadena con la operación y el resultado de la multiplicación y la añade como un nuevo elemento a la colección `Items` del `ListBox`.
    - **Utilización:** Es el componente principal para presentar visualmente la tabla de multiplicar al usuario.

Programa 59. Aplicación que genera contraseñas aleatorias

- **Explicación del Ejercicio:** Este programa tiene como objetivo **generar una contraseña aleatoria** de una longitud fija (en este caso, 8 caracteres). La contraseña se construye

combinando letras mayúsculas, minúsculas, números y símbolos seleccionados al azar de un conjunto predefinido. La contraseña generada se muestra en un `Label`.

- **Controles Necesarios:**
  - **Button:**
    - **Nombre (Name):** `btnGenerar`. Identificador del botón que activará la generación de la contraseña.
    - **Texto (Text):** "Generar Contraseña". Texto visible en el botón.
    - **Evento `Click`:** `btnGenerar_Click`. Se ejecuta cuando el usuario hace clic en el botón. Contiene toda la lógica para construir la contraseña aleatoria.
    - **Utilización:** El botón sirve como el disparador para generar una nueva contraseña.
  - **Label:**
    - **Nombre (Name):** `lblPassword`. Identificador de la etiqueta.
    - **Propiedad `Text`:** Se utiliza para mostrar la contraseña aleatoria generada una vez que el proceso ha finalizado (`lblPassword.Text = sb.ToString()`).
    - **Utilización:** Presenta la contraseña generada al usuario.
  - **Variables y Clases de Soporte (No controles visuales, pero esenciales):**
    - **`chars` (String):** Una variable de cadena que contiene todos los caracteres posibles (letras, números, símbolos) que pueden formar parte de la contraseña.
    - **`Random` (`rnd`):** Una instancia de la clase `Random` se crea (`Dim rnd As New Random()`) a nivel de clase para generar números pseudoaleatorios. `rnd.Next(chars.Length)` se utiliza para obtener un índice aleatorio dentro de la cadena `chars`, seleccionando así un carácter al azar.
    - **`StringBuilder` (`sb`):** Se importa el espacio de nombres `System.Text` para utilizar esta clase (`Dim sb As New StringBuilder()`). `StringBuilder` es una clase más eficiente para manipular y construir cadenas de texto repetidamente dentro de un bucle, ya que evita la creación de múltiples objetos `String` intermedios. `sb.Append()` se usa para añadir cada carácter aleatorio a la contraseña.
    - **Bucle `For`:** Se itera 8 veces para construir una contraseña de 8 caracteres. En cada iteración, se selecciona un carácter aleatorio de `chars` y se añade a `sb`.

Programa 60. Aplicación que muestra mensajes aleatorios

- **Explicación del Ejercicio:** Este programa tiene como función principal **mostrar un mensaje aleatorio** al usuario cada vez que presiona un botón. Los mensajes se eligen de una lista predefinida (un array de cadenas).
- **Controles Necesarios:**
  - **Button:**
    - **Nombre (Name):** `btnMensaje`. Identificador del botón.
    - **Texto (Text):** "Mostrar Mensaje". Texto visible en la superficie del botón.

        - **Evento `Click`:** `btnMensaje_Click`. Se dispara cuando el usuario hace clic. Contiene la lógica para seleccionar un mensaje aleatorio del array y mostrarlo en el `Label`.
        - **Utilización:** El botón es el disparador para obtener y mostrar un nuevo mensaje aleatorio.
    - **Label:**
        - **Nombre (Name):** `lblMensaje`. Identificador de la etiqueta.
        - **Propiedad `Text`:** Se utiliza para mostrar el mensaje aleatorio seleccionado del array (`lblMensaje.Text = mensajes(index)`).
        - **Utilización:** Presenta el mensaje aleatorio al usuario.
    - **Variables de Soporte (No controles visuales):**
        - **`mensajes()` (Array de String):** Se declara un array de cadenas a nivel de clase (`Dim mensajes() As String = {"Hola!", ...}`). Este array almacena todos los mensajes posibles que el programa puede mostrar. Declararlo a nivel de clase asegura que los mensajes estén disponibles durante toda la vida del formulario.
        - **`Random` (`rnd`):** Una instancia de la clase `Random` se crea (`Dim rnd As New Random()`) dentro del evento `Click`. `rnd.Next(mensajes.Length)` se utiliza para generar un número entero aleatorio que servirá como índice para seleccionar uno de los mensajes del array `mensajes`. El `Length` del array define el límite superior (exclusivo) para la generación del número aleatorio.
        - **Utilización:** `mensajes()` guarda los textos a mostrar; `rnd` elige cuál de ellos se mostrará.

Programa 61. Aplicación que calcula el IMC

- **Explicación del Ejercicio:** Este programa permite al usuario **calcular el Índice de Masa Corporal (IMC)** a partir de su peso en kilogramos y su altura en metros. Muestra el resultado en una etiqueta, validando que la altura sea un valor positivo para evitar errores matemáticos.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtPeso`. Es el control de entrada para el peso del usuario.
        - **Propiedad `Placeholder`:** "Peso (kg)". Proporciona una sugerencia visual al usuario sobre qué tipo de dato debe ingresar en el campo.
        - **Propiedad `Text`:** Almacena el valor del peso introducido por el usuario. Su contenido se convierte a un valor numérico de tipo `Double` utilizando la función `Val()` para realizar el cálculo.
        - **Utilización:** Permite al usuario introducir su peso en kilogramos.
    - **TextBox:**
        - **Nombre (Name):** `txtAltura`. Es el control de entrada para la altura del usuario.

        - **Propiedad `Placeholder`:** "Altura (m)". Indica al usuario que debe introducir la altura en metros.
        - **Propiedad `Text`:** Almacena el valor de la altura. Su contenido también se convierte a `Double` con `Val()`. Es crucial que este valor sea validado para ser mayor que cero (`altura > 0`) antes del cálculo, para prevenir una división por cero, lo que causaría un error en la aplicación.
        - **Utilización:** Permite al usuario introducir su altura en metros.
    - **Button:**
        - **Nombre (Name):** `btnCalcular`.
        - **Propiedad `Text`:** "Calcular IMC". Texto visible en el botón que describe su acción.
        - **Evento `Click`:** `btnCalcular_Click`. Este evento se dispara cuando el usuario presiona el botón. Contiene la lógica para obtener los valores de peso y altura, realizar el cálculo del IMC y mostrar el resultado.
        - **Utilización:** Es el disparador para iniciar el cálculo del IMC.
    - **Label:**
        - **Nombre (Name):** `lblResultado`.
        - **Propiedad `Text`:** Se actualiza para mostrar el resultado del IMC, o un mensaje de error si la altura es inválida (por ejemplo, "Altura inválida"). El resultado numérico del IMC se formatea a dos decimales (`imc.ToString("0.00")`) para una mejor presentación.
        - **Utilización:** Presenta el resultado del cálculo o mensajes informativos al usuario.

Programa 62. Conversor de unidades de longitud (metros a cm, mm, km)

- **Explicación del Ejercicio:** Este programa proporciona una herramienta para **convertir una longitud dada en metros** a sus equivalentes en centímetros, milímetros y kilómetros. Los resultados se presentan en una lista (`ListBox`).
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtMetros`. Campo de entrada para la longitud en metros.
        - **Propiedad `Text`:** Contiene el valor numérico en metros que el usuario desea convertir. Se convierte a `Double` usando `Val()` antes de realizar las operaciones.
        - **Utilización:** Permite al usuario ingresar la longitud en metros.
    - **Button:**
        - **Nombre (Name):** `btnConvertir`.
        - **Propiedad `Text`:** "Convertir". Texto visible en el botón.
        - **Evento `Click`:** `btnConvertir_Click`. Este evento se activa al presionar el botón. Antes de realizar nuevas conversiones, limpia la lista de resultados (`lstResultados.Items.Clear()`) y luego calcula y añade cada conversión a la lista.
        - **Utilización:** Inicia el proceso de conversión de unidades.

- **ListBox:**
  - **Nombre (Name):** `lstResultados`. Control para mostrar los resultados de las conversiones.
  - **Colección `Items`:** Almacena las cadenas de texto que representan cada conversión realizada.
  - **Método `Items.Clear()`:** Se llama para eliminar los resultados de conversiones anteriores, asegurando que solo se muestre el cálculo actual.
  - **Método `Items.Add()`:** Se utiliza para añadir cada resultado de conversión (metros a centímetros, milímetros, kilómetros) como una nueva línea en el `ListBox`.
  - **Utilización:** Muestra de forma organizada los resultados de las conversiones al usuario.

Programa 63. Conversor de unidades de peso (kg a g, lb)

- **Explicación del Ejercicio:** Este programa permite **convertir una cantidad de peso expresada en kilogramos** a sus equivalentes en gramos y libras. Los resultados de estas conversiones se muestran en un `ListBox`.
- **Controles Necesarios:**
  - **TextBox:**
    - **Nombre (Name):** `txtKg`. Campo de entrada para el peso en kilogramos.
    - **Propiedad `Text`:** Almacena el valor numérico en kilogramos. Se convierte a `Double` con `Val()` para los cálculos.
    - **Utilización:** Permite al usuario introducir el peso en kilogramos.
  - **Button:**
    - **Nombre (Name):** `btnConvertir`.
    - **Propiedad `Text`:** "Convertir". Texto visible en el botón.
    - **Evento `Click`:** `btnConvertir_Click`. Se dispara al hacer clic. Limpia la lista de resultados y luego realiza las conversiones para gramos y libras, añadiéndolas a la lista.
    - **Utilización:** Inicia el proceso de conversión de unidades de peso.
  - **ListBox:**
    - **Nombre (Name):** `lstResultados`. Control para mostrar los resultados de las conversiones.
    - **Colección `Items`:** Almacena los resultados de las conversiones.
    - **Método `Items.Clear()`:** Limpia los resultados anteriores antes de mostrar los nuevos.
    - **Método `Items.Add()`:** Añade cada línea de conversión (kg a gramos y kg a libras) al `ListBox`.
    - **Utilización:** Muestra las equivalencias de peso de forma estructurada.

Programa 64. Conversor de unidades de tiempo (horas a minutos y segundos)

- **Explicación del Ejercicio:** Este programa es una herramienta sencilla para **convertir una cantidad de tiempo dada en horas** a sus equivalentes en minutos y segundos. El resultado se muestra en una etiqueta.

- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtHoras`. Campo de entrada para la cantidad de horas.
        - **Propiedad `Text`:** Contiene el valor numérico en horas. Se convierte a `Double` con `Val()` para los cálculos.
        - **Utilización:** Permite al usuario ingresar la cantidad de horas.
    - **Button:**
        - **Nombre (Name):** `btnConvertir`.
        - **Propiedad `Text`:** "Convertir". Texto visible en el botón.
        - **Evento `Click`:** `btnConvertir_Click`. Se activa al hacer clic. Obtiene el valor en horas, realiza las conversiones y actualiza la etiqueta de resultado.
        - **Utilización:** Desencadena la conversión de unidades de tiempo.
    - **Label:**
        - **Nombre (Name):** `lblResultado`.
        - **Propiedad `Text`:** Muestra el resultado de la conversión en un formato legible, por ejemplo: "5 h = 300 min = 18000 seg".
        - **Utilización:** Presenta al usuario la equivalencia de horas en minutos y segundos.

Programa 65. Conversor de unidades de velocidad (km/h a m/s)

- **Explicación del Ejercicio:** Este programa convierte una velocidad ingresada en **kilómetros por hora (km/h)** a su equivalente en **metros por segundo (m/s)**. El resultado se muestra en una etiqueta, formateado a dos decimales.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtKmH`. Campo de entrada para la velocidad en km/h.
        - **Propiedad `Text`:** Contiene el valor numérico de la velocidad. Se convierte a `Double` utilizando `Val()` para la operación.
        - **Utilización:** Permite al usuario introducir la velocidad en kilómetros por hora.
    - **Button:**
        - **Nombre (Name):** `btnConvertir`.
        - **Propiedad `Text`:** "Convertir". Texto visible en el botón.
        - **Evento `Click`:** `btnConvertir_Click`. Se activa al hacer clic. Obtiene la velocidad en km/h, realiza la conversión y actualiza la etiqueta de resultado.
        - **Utilización:** Inicia la conversión de velocidad.
    - **Label:**
        - **Nombre (Name):** `lblResultado`.
        - **Propiedad `Text`:** Muestra la velocidad original en km/h y su equivalente en m/s. El resultado de m/s se formatea a dos decimales (`ms.ToString("0.00")`) para mayor precisión y legibilidad.
        - **Utilización:** Presenta el resultado de la conversión al usuario.

Programa 66. Aplicación que guarda configuraciones en My.Settings

- **Explicación del Ejercicio:** Este programa demuestra cómo **guardar y cargar una configuración de usuario (una cadena de texto)** de forma persistente entre las ejecuciones de la aplicación utilizando la funcionalidad `My.Settings` de Visual Basic .NET. Al guardar, el dato se mantiene incluso si la aplicación se cierra y se vuelve a abrir.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtDato`.
        - **Propiedad `Text`:** Contiene el dato (texto) que el usuario desea guardar o que se ha cargado previamente.
        - **Utilización:** Es el campo donde el usuario introduce o visualiza el dato de configuración.
    - **Button:**
        - **Nombre (Name):** `btnGuardar`.
        - **Propiedad `Text`:** "Guardar". Texto visible en el botón.
        - **Evento `Click`:** `btnGuardar_Click`. Al hacer clic, el valor de `txtDato.Text` se asigna a la propiedad `My.Settings.MiDato`. Inmediatamente después, se llama a `My.Settings.Save()` para escribir este cambio en el disco y asegurar que se persista.
        - **Utilización:** Permite al usuario guardar el dato de configuración.
    - **Button:**
        - **Nombre (Name):** `btnCargar`.
        - **Propiedad `Text`:** "Cargar". Texto visible en el botón.
        - **Evento `Click`:** `btnCargar_Click`. Al hacer clic, recupera el valor almacenado en `My.Settings.MiDato` y lo asigna a `lblResultado.Text` para mostrarlo al usuario.
        - **Utilización:** Permite al usuario cargar el último dato guardado en la configuración.
    - **Label:**
        - **Nombre (Name):** `lblResultado`.
        - **Propiedad `Text`:** Muestra el dato que se ha cargado de `My.Settings` (ej., "Dato guardado: [valor]").
        - **Utilización:** Informa al usuario sobre el valor de la configuración cargada.
- **Configuración Adicional (en Visual Studio Designer):**
    - **`My.Settings`:** Para que este programa funcione, debe agregarse una nueva configuración en la sección `My Project > Settings`. Esta configuración debe llamarse **`MiDato`**, ser de tipo **`String`** y tener un alcance de **`User`**. El alcance `User` es crucial para que la configuración se guarde y cargue por usuario.

Programa 67. Calculadora de área de figuras geométricas (círculo y cuadrado)

- **Explicación del Ejercicio:** Este programa calcula el **área de un círculo y el área de un cuadrado**. El usuario introduce el radio del círculo y el lado del cuadrado en campos de texto separados, y al presionar un botón, se muestran ambos resultados en una etiqueta.
- **Controles Necesarios:**

        - **TextBox:**
            - **Nombre (Name):** `txtRadio`. Campo de entrada para el radio del círculo.
            - **Propiedad `Text`:** Contiene el valor numérico del radio. Se convierte a `Double` usando `Val()` para los cálculos.
            - **Utilización:** Permite al usuario ingresar el radio del círculo.
        - **TextBox:**
            - **Nombre (Name):** `txtLado`. Campo de entrada para el lado del cuadrado.
            - **Propiedad `Text`:** Contiene el valor numérico del lado. Se convierte a `Double` usando `Val()` para los cálculos.
            - **Utilización:** Permite al usuario ingresar el lado del cuadrado.
        - **Button:**
            - **Nombre (Name):** `btnCalcular`.
            - **Propiedad `Text`:** "Calcular Áreas". Texto visible en el botón.
            - **Evento `Click`:** `btnCalcular_Click`. Se activa al hacer clic. Obtiene los valores de radio y lado, aplica las fórmulas de área y muestra los resultados en la etiqueta.
            - **Utilización:** Inicia el cálculo de las áreas.
        - **Label:**
            - **Nombre (Name):** `lblResultado`.
            - **Propiedad `Text`:** Muestra el área calculada del círculo (formateada a dos decimales con `ToString("0.00")`) y el área del cuadrado, separadas por un salto de línea (`vbCrLf`).
            - **Utilización:** Presenta los resultados de las áreas calculadas al usuario.
    - **Funciones y Métodos Adicionales:**
        - **`Math.PI`:** Constante estática de la clase `Math` que proporciona el valor de Pi, utilizado en el cálculo del área del círculo.
        - **Operador `^`:** Se utiliza para elevar el radio al cuadrado (`radio ^ 2`).

Programa 68. Calculadora de perímetro de figuras geométricas (círculo y cuadrado)

- **Explicación del Ejercicio:** Este programa calcula el **perímetro de un círculo y el perímetro de un cuadrado**, de forma similar al ejercicio anterior para áreas. El usuario ingresa el radio y el lado, y el programa muestra ambos resultados.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtRadio`. Campo de entrada para el radio del círculo.
        - **Propiedad `Text`:** Contiene el valor numérico del radio. Se convierte a `Double` usando `Val()` para los cálculos.
        - **Utilización:** Permite al usuario ingresar el radio del círculo.
    - **TextBox:**
        - **Nombre (Name):** `txtLado`. Campo de entrada para el lado del cuadrado.
        - **Propiedad `Text`:** Contiene el valor numérico del lado. Se convierte a `Double` usando `Val()` para los cálculos.
        - **Utilización:** Permite al usuario ingresar el lado del cuadrado.
    - **Button:**

        - **Nombre (Name):** `btnCalcular`.
        - **Propiedad `Text`:** "Calcular Perímetros". Texto visible en el botón.
        - **Evento `Click`:** `btnCalcular_Click`. Se activa al hacer clic. Obtiene los valores de radio y lado, aplica las fórmulas de perímetro y muestra los resultados en la etiqueta.
        - **Utilización:** Inicia el cálculo de los perímetros.
    - **Label:**
        - **Nombre (Name):** `lblResultado`.
        - **Propiedad `Text`:** Muestra el perímetro calculado del círculo (formateado a dos decimales) y el perímetro del cuadrado, separadas por un salto de línea (`vbCrLf`).
        - **Utilización:** Presenta los resultados de los perímetros calculados al usuario.
- **Funciones y Métodos Adicionales:**
    - **`Math.PI`:** Constante estática de la clase `Math` que proporciona el valor de Pi, utilizado en el cálculo del perímetro del círculo.

Programa 69. Juego del número secreto

- **Explicación del Ejercicio:** Este programa implementa un juego simple de adivinanza. La aplicación **genera un número secreto aleatorio** entre 1 y 100. El usuario introduce intentos en un cuadro de texto, y el programa le indica si el número secreto es mayor, menor o si ha acertado.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtIntento`. Campo de entrada donde el usuario escribe su número para adivinar.
        - **Propiedad `Text`:** Contiene el valor numérico del intento del usuario. Se convierte a `Integer` con `Val()` para la comparación.
        - **Utilización:** Es la interfaz para que el usuario ingrese sus suposiciones.
    - **Button:**
        - **Nombre (Name):** `btnProbar`.
        - **Propiedad `Text`:** "Probar". Texto visible en el botón.
        - **Evento `Click`:** `btnProbar_Click`. Se activa al hacer clic. Obtiene el intento del usuario, lo compara con el `secreto` y actualiza el `lblResultado` con una pista o el mensaje de victoria.
        - **Utilización:** Desencadena la verificación del intento del usuario.
    - **Label:**
        - **Nombre (Name):** `lblResultado`.
        - **Propiedad `Text`:** Muestra mensajes al usuario como "¡Correcto!", "El número es mayor" o "El número es menor", guiando al jugador o anunciando la victoria.
        - **Utilización:** Proporciona retroalimentación visual al usuario sobre su progreso en el juego.
- **Variables y Clases de Soporte (No controles visuales):**

- **`secreto` (Integer):** Variable a nivel de clase que almacena el número aleatorio que el usuario debe adivinar.
- **`Random` (`rnd`):** Una instancia de esta clase, declarada a nivel de clase, se utiliza para generar el número aleatorio inicial.
- **`Form1_Load` (Evento):** Este evento se dispara cuando el formulario se carga por primera vez. Aquí, `secreto` se inicializa con un valor aleatorio entre 1 y 100 utilizando `rnd.Next(1, 101)`.

Programa 70. Juego de adivinar palabra

- **Explicación del Ejercicio:** Este programa implementa un **juego de adivinanza de palabras** simple. Hay una palabra oculta predefinida, y el usuario intenta adivinarla letra por letra. Si una letra es correcta, se revela en la posición correspondiente; si se adivina toda la palabra, el usuario gana.
- **Controles Necesarios:**
  - **Label:**
    - **Nombre (Name):** `lblPista`. Inicialmente muestra la palabra oculta con guiones bajos (ej., "Palabra: _ _ _ _").
    - **Propiedad `Text`:** Se actualiza con el progreso de la adivinanza, mostrando las letras correctas que el usuario ha encontrado, o la palabra completa al ganar.
    - **Utilización:** Muestra visualmente la palabra a adivinar y el progreso del jugador.
  - **TextBox:**
    - **Nombre (Name):** `txtLetra`. Campo de entrada donde el usuario introduce una letra para probar.
    - **Propiedad `Text`:** Contiene la letra ingresada por el usuario. Su contenido se convierte a mayúsculas con `ToUpper()` para una comparación sin distinción de mayúsculas/minúsculas.
    - **Método `Clear()`:** Se utiliza para vaciar el `TextBox` después de cada intento, listo para la siguiente letra.
    - **Utilización:** Permite al usuario ingresar una letra para adivinar.
  - **Button:**
    - **Nombre (Name):** `btnProbar`.
    - **Propiedad `Text`:** "Probar Letra". Texto visible en el botón.
    - **Evento `Click`:** `btnProbar_Click`. Se activa al hacer clic. Contiene la lógica principal del juego: procesa la letra ingresada, verifica si está en la palabra secreta, actualiza la `lblPista` y comprueba si el juego ha terminado (victoria).
    - **Utilización:** Inicia la acción de probar una letra.
  - **Label:**
    - **Nombre (Name):** `lblResultado`.
    - **Propiedad `Text`:** Muestra un mensaje de "¡Ganaste!" cuando el usuario adivina la palabra completa.
    - **Utilización:** Notifica la victoria al usuario.

- **Variables y Clases de Soporte (No controles visuales):**
    - **`palabra` (String):** Almacena la palabra secreta a adivinar (ej., "GATO"). Se declara a nivel de clase.
    - **`adivinada` (String):** Almacena la representación de la palabra con guiones bajos, que se va revelando a medida que el usuario acierta letras (ej., "____"). Se declara a nivel de clase.
    - **`Form1_Load` (Evento):** Se ejecuta al cargar el formulario y establece el texto inicial de `lblPista` con la palabra oculta (`adivinada`).
    - **`letra.Length = 1` (Validación):** Asegura que el usuario solo haya ingresado una única letra.
    - **`.ToCharArray()`:** Convierte la cadena `adivinada` en un array de caracteres (`nueva`) para que los caracteres individuales puedan modificarse.
    - **`New String(nueva)`:** Una vez modificado el array de caracteres, se utiliza para construir una nueva cadena `adivinada` a partir del array.
    - **`For i As Integer = 0 To palabra.Length - 1` (Bucle):** Itera sobre cada carácter de la palabra secreta para buscar coincidencias con la letra ingresada.

Programa 71. Juego de piedra, papel o tijera

- **Explicación del Ejercicio:** Este programa simula el clásico juego de **piedra, papel o tijera** entre el usuario y la computadora. El usuario elige su jugada desde una lista desplegable, y la computadora hace una elección aleatoria. El programa determina el ganador y muestra el resultado.
- **Controles Necesarios:**
    - **ComboBox:**
        - **Nombre (Name):** `cmbJugador`. Proporciona las opciones de jugada para el usuario.
        - **Colección `Items`:** Se pobla con las cadenas "Piedra", "Papel", "Tijera".
        - **Propiedad `Text`:** Contiene la elección actual del jugador. Se verifica si está vacía para asegurar que el usuario ha seleccionado una opción.
        - **Utilización:** Permite al usuario seleccionar su jugada de una lista predefinida.
    - **Button:**
        - **Nombre (Name):** `btnJugar`.
        - **Propiedad `Text`:** "Jugar". Texto visible en el botón.
        - **Evento `Click`:** `btnJugar_Click`. Se activa al hacer clic. Obtiene la elección del jugador y genera la elección aleatoria de la computadora. Luego, aplica la lógica del juego para determinar el ganador y actualiza el `lblResultado`.
        - **Utilización:** Inicia una ronda del juego.
    - **Label:**
        - **Nombre (Name):** `lblResultado`.
        - **Propiedad `Text`:** Muestra el resultado de la ronda (por ejemplo, "Ganaste", "Perdiste", "Empate"), indicando también la elección de la computadora (ej., "Empate, PC eligió Papel").
        - **Utilización:** Informa al usuario sobre el desarrollo y el resultado del juego.

- **Variables y Clases de Soporte (No controles visuales):**
  - **`opciones()` (Array de String):** Almacena las tres posibles jugadas del juego ("Piedra", "Papel", "Tijera"). Declarado a nivel de clase.
  - **`Random` (`rnd`):** Una instancia de esta clase, declarada a nivel de clase, se utiliza para generar la elección aleatoria de la computadora (`rnd.Next(opciones.Length)`).

Programa 72. Cronómetro con Start, Stop y Reset

- **Explicación del Ejercicio:** Este programa implementa un **cronómetro básico** con las funcionalidades esenciales: iniciar, detener y reiniciar. El tiempo transcurrido se muestra en una etiqueta en formato de horas, minutos y segundos.
- **Controles Necesarios:**
  - **Label:**
    - **Nombre (Name):** `lblTiempo`. Inicialmente muestra "00:00:00".
    - **Propiedad `Text`:** Se actualiza cada segundo para mostrar el tiempo transcurrido, formateado como "hh:mm:ss" utilizando `TimeSpan.FromSeconds()` y `.ToString("hh\:mm\:ss")`.
    - **Utilización:** Muestra el tiempo actual del cronómetro.
  - **Button:**
    - **Nombre (Name):** `btnStart`.
    - **Propiedad `Text`:** "Start". Texto visible en el botón.
    - **Evento `Click`:** `btnStart_Click`. Llama al método `Timer1.Start()` para comenzar la cuenta del cronómetro.
    - **Utilización:** Inicia o reanuda el cronómetro.
  - **Button:**
    - **Nombre (Name):** `btnStop`.
    - **Propiedad `Text`:** "Stop". Texto visible en el botón.
    - **Evento `Click`:** `btnStop_Click`. Llama al método `Timer1.Stop()` para pausar el cronómetro.
    - **Utilización:** Detiene o pausa el cronómetro.
  - **Button:**
    - **Nombre (Name):** `btnReset`.
    - **Propiedad `Text`:** "Reset". Texto visible en el botón.
    - **Evento `Click`:** `btnReset_Click`. Detiene el `Timer1`, reinicia la variable `segundos` a 0 y restablece el `lblTiempo.Text` a "00:00:00".
    - **Utilización:** Reinicia el cronómetro a cero.
  - **Timer:**
    - **Nombre (Name):** `Timer1`. Componente no visible que genera eventos en intervalos regulares.
    - **Propiedad `Interval`:** 1000. Se configura para disparar su evento `Tick` cada 1000 milisegundos (1 segundo).
    - **Evento `Tick`:** `Timer1_Tick`. En cada disparo, incrementa la variable `segundos` en 1 y actualiza la propiedad `Text` de `lblTiempo` con el nuevo tiempo formateado.

        - **Utilización:** Es el motor del cronómetro, responsable de la actualización del tiempo.
- **Variables y Clases de Soporte (No controles visuales):**
    - **`segundos` (Integer):** Variable a nivel de clase que acumula el número total de segundos transcurridos desde el último reinicio.
    - **`TimeSpan`:** La clase `TimeSpan` se utiliza para representar intervalos de tiempo. `TimeSpan.FromSeconds()` crea un objeto `TimeSpan` a partir de un número de segundos, que luego puede formatearse.

Programa 73. Temporizador con alarma

- **Explicación del Ejercicio:** Este programa permite al usuario establecer un **temporizador de cuenta regresiva** en segundos. Cuando el tiempo llega a cero, el programa detiene la cuenta y muestra un mensaje de alarma en un `MessageBox`.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtSegundos`. Campo de entrada donde el usuario especifica la cantidad de segundos para el temporizador.
        - **Propiedad `Placeholder`:** "Segundos". Sugerencia visual para el usuario.
        - **Propiedad `Text`:** Contiene el valor numérico de los segundos. Se convierte a `Integer` usando `Val()` cuando se inicia el temporizador.
        - **Utilización:** Permite al usuario definir la duración del temporizador.
    - **Button:**
        - **Nombre (Name):** `btnIniciar`.
        - **Propiedad `Text`:** "Iniciar". Texto visible en el botón.
        - **Evento `Click`:** `btnIniciar_Click`. Se activa al hacer clic. Obtiene el valor de `txtSegundos.Text`, lo asigna a la variable `tiempo` y luego llama a `Timer1.Start()` para iniciar el temporizador.
        - **Utilización:** Comienza la cuenta regresiva.
    - **Label:**
        - **Nombre (Name):** `lblTiempo`.
        - **Propiedad `Text`:** Muestra los segundos restantes de la cuenta regresiva (ej., "10 s", "9 s"). Se actualiza en cada `Tick` del `Timer`.
        - **Utilización:** Proporciona una visualización en tiempo real de los segundos que quedan.
    - **Timer:**
        - **Nombre (Name):** `Timer1`. Componente no visible.
        - **Propiedad `Interval`:** 1000. Se configura para disparar su evento `Tick` cada 1000 milisegundos (1 segundo).
        - **Evento `Tick`:** `Timer1_Tick`. En cada disparo, decrementa la variable `tiempo` en 1. Si `tiempo` aún es mayor que 0, actualiza `lblTiempo`. Si `tiempo` llega a 0, detiene el `Timer1` y muestra un `MessageBox` con el mensaje de alarma "¡Tiempo terminado!".
        - **Utilización:** Es el motor de la cuenta regresiva y el disparador de la alarma.
- **Variables de Soporte (No controles visuales):**

- **`tiempo` (Integer):** Variable a nivel de clase que almacena la cantidad de segundos que quedan en la cuenta regresiva.

Programa 74. Editor de listas (agregar, modificar y borrar)

- **Explicación del Ejercicio:** Este programa funciona como un **editor básico para una lista de elementos (`ListBox`)**, permitiendo al usuario realizar tres operaciones fundamentales: **agregar** nuevos elementos, **modificar** un elemento ya seleccionado y **borrar** un elemento seleccionado. Es una funcionalidad común en aplicaciones de gestión de datos.
- **Controles Necesarios:**
  - **TextBox:**
    - **Nombre (Name):** `txtElemento`. Campo de entrada para escribir el texto de los elementos de la lista.
    - **Propiedad `Text`:** Contiene el texto que se agregará o que se usará para modificar un elemento existente.
    - **Método `Clear()`:** Se utiliza después de agregar un elemento para limpiar el campo y prepararlo para la siguiente entrada.
    - **Utilización:** Interfaz para introducir el texto de los elementos.
  - **Button:**
    - **Nombre (Name):** `btnAgregar`.
    - **Propiedad `Text`:** "Agregar". Texto visible en el botón.
    - **Evento `Click`:** `btnAgregar_Click`. Al hacer clic, toma el texto de `txtElemento.Text` y lo añade como un nuevo elemento a la colección `Items` de `lstElementos`. Después, limpia `txtElemento`.
    - **Utilización:** Añade un nuevo elemento a la lista.
  - **Button:**
    - **Nombre (Name):** `btnModificar`.
    - **Propiedad `Text`:** "Modificar". Texto visible en el botón.
    - **Evento `Click`:** `btnModificar_Click`. Se activa al hacer clic. Primero, verifica si hay un elemento seleccionado en `lstElementos` (`lstElementos.SelectedIndex >= 0`). Si hay una selección, reemplaza el texto del elemento seleccionado con el contenido actual de `txtElemento.Text`.
    - **Utilización:** Edita el elemento de la lista que está actualmente seleccionado.
  - **Button:**
    - **Nombre (Name):** `btnBorrar`.
    - **Propiedad `Text`:** "Borrar". Texto visible en el botón.
    - **Evento `Click`:** `btnBorrar_Click`. Se activa al hacer clic. Similar a `btnModificar`, verifica si hay un elemento seleccionado. Si lo hay, utiliza `lstElementos.Items.RemoveAt(lstElementos.SelectedIndex)` para eliminar el elemento en esa posición.
    - **Utilización:** Elimina el elemento de la lista que está actualmente seleccionado.
  - **ListBox:**

        - **Nombre (Name):** `lstElementos`. Es el control principal donde se muestran y gestionan los elementos de la lista.
        - **Colección `Items`:** Almacena todos los elementos (cadenas de texto) que son visibles en la lista.
        - **Propiedad `SelectedIndex`:** Devuelve el índice (posición) del elemento que está actualmente seleccionado por el usuario. Su valor es -1 si no hay ningún elemento seleccionado. Es fundamental para `btnModificar` y `btnBorrar`.
        - **Utilización:** Muestra y permite la interacción con la lista de elementos.

Programa 75. Editor de texto con guardar y abrir archivo

- **Explicación del Ejercicio:** Este programa simula un **editor de texto básico**, proporcionando la funcionalidad de **abrir un archivo de texto** existente para ver o editar su contenido y **guardar el texto editado** en un nuevo archivo (o sobrescribir uno existente). La interacción para abrir y guardar se realiza a través de un `MenuStrip` y los cuadros de diálogo estándar de Windows (`OpenFileDialog` y `SaveFileDialog`).
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtContenido`. Es el área principal de edición de texto.
        - **Propiedad `Multiline`:** `True`. Permite que el `TextBox` acepte y muestre múltiples líneas de texto, lo que es esencial para un editor de texto.
        - **Propiedad `Dock`:** `Fill`. Configura el `TextBox` para que se ajuste y ocupe todo el espacio disponible en el formulario, creando una interfaz similar a un bloc de notas.
        - **Propiedad `Text`:** Almacena el texto que el usuario está editando o el texto cargado de un archivo.
        - **Utilización:** El lienzo principal donde el usuario escribe, edita y visualiza el texto.
    - **MenuStrip:**
        - **Nombre (Name):** `MenuStrip1` (implícito, el código de la fuente no lo nombra explícitamente pero se asume su existencia). Se coloca en la parte superior del formulario para albergar las opciones de menú.
        - **Elementos de Menú:** Contiene "Archivo → Abrir" y "Archivo → Guardar".
        - **Utilización:** Proporciona un menú desplegable para las operaciones de archivo.
    - **ToolStripMenuItem (Abrir):**
        - **Evento `Click`:** `AbrirToolStripMenuItem_Click`. Este evento se dispara cuando el usuario selecciona la opción "Abrir" del menú. Abre un `OpenFileDialog`, y si el usuario selecciona un archivo (`DialogResult.OK`), utiliza `System.IO.File.ReadAllText()` para leer todo el contenido del archivo y lo asigna a `txtContenido.Text`.
        - **Utilización:** Permite al usuario cargar el contenido de un archivo de texto existente.

    - **ToolStripMenuItem (Guardar):**
        - **Evento `Click`:** `GuardarToolStripMenuItem_Click`. Se dispara cuando el usuario selecciona la opción "Guardar" del menú. Abre un `SaveFileDialog`, y si el usuario confirma la ubicación y nombre del archivo (`DialogResult.OK`), utiliza `System.IO.File.WriteAllText()` para guardar el contenido de `txtContenido.Text` en el archivo especificado.
        - **Utilización:** Permite al usuario guardar el texto editado en un archivo.
    - **OpenFileDialog:**
        - **Nombre (Name):** `OpenFileDialog1`. Es un componente no visible que muestra el cuadro de diálogo estándar de Windows para seleccionar un archivo.
        - **Método `ShowDialog()`:** Muestra el cuadro de diálogo y retorna un `DialogResult` (`OK` si el usuario hace clic en "Abrir").
        - **Propiedad `FileName`:** Contiene la ruta completa del archivo que el usuario ha seleccionado para abrir.
        - **Utilización:** Facilita la navegación del usuario para elegir un archivo de texto.
    - **SaveFileDialog:**
        - **Nombre (Name):** `SaveFileDialog1`. Es un componente no visible que muestra el cuadro de diálogo estándar de Windows para guardar un archivo.
        - **Método `ShowDialog()`:** Muestra el cuadro de diálogo y retorna un `DialogResult` (`OK` si el usuario hace clic en "Guardar").
        - **Propiedad `FileName`:** Contiene la ruta completa (incluyendo el nombre del archivo) donde el usuario desea guardar el contenido.
        - **Utilización:** Facilita la navegación del usuario para especificar la ubicación y nombre del archivo a guardar.
- **Importación de Namespace:**
    - **`Imports System.IO`:** Necesario para utilizar los métodos de lectura y escritura de archivos (`File.ReadAllText`, `File.WriteAllText`).

Programa 76. Aplicación con varios formularios enlazados

- **Explicación del Ejercicio:** Este programa ilustra un concepto fundamental en aplicaciones de interfaz gráfica: cómo una aplicación puede tener **múltiples ventanas o formularios** y cómo un formulario principal (`Form1`) puede **abrir y mostrar otra ventana secundaria** (`Form2`).
- **Controles Necesarios:**
    - **Button (en `Form1`):**
        - **Nombre (Name):** `btnAbrir`.
        - **Propiedad `Text`:** "Abrir Form2". Texto visible en el botón que indica su función.
        - **Evento `Click`:** `btnAbrir_Click`. Este evento se dispara cuando el usuario hace clic en el botón. Dentro de este evento, se crea una nueva instancia de `Form2` y se invoca su método `Show()` para hacerla visible.

        - **Utilización:** Actúa como el disparador para abrir la ventana secundaria.
    - **Formulario `Form2` (Formulario Adicional):**
        - **Requisito Adicional:** Para este ejercicio, se debe crear un **segundo formulario** en el proyecto de Visual Basic .NET. Se le puede dar el nombre predeterminado `Form2` y puede estar completamente vacío o contener algún control simple como una etiqueta para indicar que es el nuevo formulario.
        - **Utilización:** Es la ventana que se abre cuando el usuario interactúa con `Form1`, demostrando la capacidad de la aplicación para gestionar múltiples interfaces.
- **Clases y Métodos de Soporte:**
    - **`Dim f2 As New Form2`:** Esta línea de código crea una **nueva instancia** de la clase `Form2`. Esto significa que se crea un nuevo objeto `Form2` en la memoria, el cual representa una ventana independiente.
    - **`f2.Show()`:** Este método hace que la instancia del formulario (`f2`) sea **visible en la pantalla**. Abre una nueva ventana de la aplicación que opera independientemente de `Form1` (aunque `Form1` sigue siendo accesible).

Programa 77. Agenda que guarda datos en archivo de texto

- **Explicación del Ejercicio:** Este programa es una **agenda simple** que permite al usuario registrar nombres y números de teléfono de contactos. La particularidad es que cada nuevo contacto se **guarda en un archivo de texto** (`agenda.txt`) y también se muestra en un `ListBox` en el formulario.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtNombre`. Campo de entrada para el nombre del contacto.
        - **Propiedad `Text`:** Contiene el nombre introducido por el usuario.
        - **Método `Clear()`:** Se utiliza para vaciar el `TextBox` después de que el contacto se ha guardado, preparándolo para la siguiente entrada.
        - **Utilización:** Permite al usuario introducir el nombre del contacto.
    - **TextBox:**
        - **Nombre (Name):** `txtTelefono`. Campo de entrada para el número de teléfono del contacto.
        - **Propiedad `Text`:** Contiene el número de teléfono.
        - **Método `Clear()`:** Se utiliza para limpiar el `TextBox` después de guardar.
        - **Utilización:** Permite al usuario introducir el número de teléfono del contacto.
    - **Button:**
        - **Nombre (Name):** `btnGuardar`.
        - **Propiedad `Text`:** "Guardar". Texto visible en el botón.
        - **Evento `Click`:** `btnGuardar_Click`. Se activa al hacer clic. Concatena el nombre y el teléfono en una sola cadena, la añade al archivo "agenda.txt" (con un salto de línea) y también a `lstContactos`. Finalmente, limpia ambos `TextBox`.
        - **Utilización:** Disparador para añadir y guardar un nuevo contacto.

        - **ListBox:**
            - **Nombre (Name):** `lstContactos`. Muestra visualmente la lista de contactos agregados.
            - **Colección `Items`:** Almacena las cadenas que representan cada contacto (ej., "Juan Pérez - 123456789").
            - **Método `Items.Add()`:** Añade la cadena del contacto a la lista visible.
            - **Utilización:** Presenta la lista de contactos de la agenda al usuario.
- **Importación de Namespace:**
    - **`Imports System.IO`:** Es necesario para utilizar los métodos de la clase `File`, como `File.AppendAllText()`.
- **Métodos de la Clase `File`:**
    - **`File.AppendAllText("agenda.txt", contacto & Environment.NewLine)`:** Este método estático es crucial. Añade la cadena `contacto` (que incluye el nombre y el teléfono) y un salto de línea (`Environment.NewLine`) al final del archivo especificado ("agenda.txt"). Si el archivo no existe, lo crea.
    - **`Environment.NewLine`:** Propiedad que proporciona la cadena de nueva línea estándar para el entorno actual, asegurando la compatibilidad del archivo de texto en diferentes sistemas operativos.

Programa 78. Agenda que guarda datos en archivo CSV

- **Explicación del Ejercicio:** Este programa es una variante de la agenda anterior. Permite al usuario registrar contactos (nombre y teléfono) y los **guarda en un archivo en formato CSV (Comma Separated Values)**. Este formato es ampliamente utilizado para intercambiar datos entre aplicaciones, especialmente hojas de cálculo. Los contactos también se muestran en un `ListBox`.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtNombre`. Campo de entrada para el nombre del contacto.
        - **Propiedad `Text`:** Contiene el nombre.
        - **Método `Clear()`:** Limpia el campo después de guardar.
        - **Utilización:** Permite introducir el nombre del contacto.
    - **TextBox:**
        - **Nombre (Name):** `txtTelefono`. Campo de entrada para el número de teléfono.
        - **Propiedad `Text`:** Contiene el teléfono.
        - **Método `Clear()`:** Limpia el campo después de guardar.
        - **Utilización:** Permite introducir el teléfono del contacto.
    - **Button:**
        - **Nombre (Name):** `btnGuardar`.
        - **Propiedad `Text`:** "Guardar". Texto visible en el botón.

        - **Evento `Click`:** `btnGuardar_Click`. Al hacer clic, formatea el nombre y el teléfono con una coma como separador, lo añade al archivo "agenda.csv" y al `ListBox`. Limpia los `TextBox`.
        - **Utilización:** Disparador para guardar el contacto en formato CSV.
    - **ListBox:**
        - **Nombre (Name):** `lstContactos`. Muestra los contactos agregados.
        - **Colección `Items`:** Almacena las cadenas formateadas como CSV.
        - **Método `Items.Add()`:** Añade el contacto formateado a la lista visible.
        - **Utilización:** Presenta la lista de contactos al usuario.
- **Importación de Namespace:**
    - **`Imports System.IO`:** Necesario para operaciones de archivo.
- **Formato de Contacto:**
    - **`Dim contacto As String = txtNombre.Text & "," & txtTelefono.Text`:** Aquí se define la clave del formato CSV, donde el nombre y el teléfono se unen con una coma.
- **Guardado en Archivo:**
    - **`File.AppendAllText("agenda.csv", contacto & Environment.NewLine)`:** Guarda la cadena `contacto` (ya formateada con comas) y un salto de línea en el archivo "agenda.csv".

Programa 79. Agenda que guarda datos en archivo XML

- **Explicación del Ejercicio:** Este programa es una extensión de las agendas anteriores, permitiendo al usuario guardar contactos (nombre y teléfono) en un **archivo XML (Extensible Markup Language)**. El XML es un formato estructurado y jerárquico que facilita la organización y el intercambio de datos. Los contactos se almacenan como elementos `Contacto` con `Nombre` y `Telefono` como atributos.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtNombre`. Campo de entrada para el nombre del contacto.
        - **Propiedad `Text`:** Contiene el nombre. Se utiliza como el valor del atributo "Nombre" en el elemento XML "Contacto".
        - **Método `Clear()`:** Limpia el `TextBox` después de guardar.
        - **Utilización:** Permite introducir el nombre del contacto.
    - **TextBox:**
        - **Nombre (Name):** `txtTelefono`. Campo de entrada para el número de teléfono.
        - **Propiedad `Text`:** Contiene el teléfono. Se utiliza como el valor del atributo "Telefono" en el elemento XML "Contacto".
        - **Método `Clear()`:** Limpia el `TextBox` después de guardar.
        - **Utilización:** Permite introducir el teléfono del contacto.
    - **Button:**
        - **Nombre (Name):** `btnGuardar`.
        - **Propiedad `Text`:** "Guardar". Texto visible en el botón.

        - **Evento `Click`:** `btnGuardar_Click`. Se activa al hacer clic. Gestiona la lógica para crear o cargar el documento XML, añadir el nuevo contacto como un elemento con atributos y luego guardar el documento XML en "agenda.xml". También añade el contacto al `ListBox`.
        - **Utilización:** Disparador para guardar el contacto en formato XML.
    - **ListBox:**
        - **Nombre (Name):** `lstContactos`. Muestra los contactos agregados en un formato legible.
        - **Colección `Items`:** Almacena las cadenas que representan cada contacto.
        - **Método `Items.Add()`:** Añade el contacto a la lista visible.
        - **Utilización:** Presenta la lista de contactos al usuario.
- **Importación de Namespace:**
    - **`Imports System.Xml`:** Es necesario para trabajar con las clases relacionadas con XML, como `XmlDocument`.
- **Clases y Métodos de XML:**
    - **`XmlDocument` (`doc`):** Una clase fundamental para representar un documento XML en memoria.
    - **`doc.Load(archivo)`:** Carga un documento XML existente desde un archivo.
    - **`doc.CreateElement("Contactos")` / `doc.CreateElement("Contacto")`:** Crea nuevos elementos XML con el nombre especificado.
    - **`doc.AppendChild(root)` / `doc.DocumentElement.AppendChild(contacto)`:** Añade un nodo (elemento) como hijo de otro nodo. `DocumentElement` se refiere al elemento raíz del documento.
    - **`contacto.SetAttribute("Nombre", txtNombre.Text)`:** Establece un atributo para el elemento XML `contacto`. Aquí, "Nombre" es el nombre del atributo y `txtNombre.Text` es su valor.
    - **`doc.Save(archivo)`:** Guarda el documento XML actual en el archivo especificado ("agenda.xml").
    - **`IO.File.Exists(archivo)`:** Se utiliza para verificar si el archivo "agenda.xml" ya existe. Si no, se crea la estructura XML inicial (el elemento raíz "Contactos").

Programa 80. Agenda que guarda datos en base de datos Access

- **Explicación del Ejercicio:** Este programa lleva la funcionalidad de la agenda un paso más allá al integrar una **base de datos Microsoft Access** para almacenar los contactos (nombre y teléfono). Esto introduce el concepto de persistencia de datos estructurada y la interacción con sistemas de gestión de bases de datos (DBMS).
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtNombre`. Campo de entrada para el nombre del contacto.

        - **Propiedad `Text`:** Contiene el nombre del contacto. Se utiliza como parámetro para la consulta SQL de inserción (`@n`).
        - **Método `Clear()`:** Limpia el `TextBox` después de guardar.
        - **Utilización:** Permite introducir el nombre del contacto a guardar.
    - **TextBox:**
        - **Nombre (Name):** `txtTelefono`. Campo de entrada para el número de teléfono del contacto.
        - **Propiedad `Text`:** Contiene el número de teléfono. Se utiliza como parámetro para la consulta SQL de inserción (`@t`).
        - **Método `Clear()`:** Limpia el `TextBox` después de guardar.
        - **Utilización:** Permite introducir el número de teléfono del contacto.
    - **Button:**
        - **Nombre (Name):** `btnGuardar`.
        - **Propiedad `Text`:** "Guardar". Texto visible en el botón.
        - **Evento `Click`:** `btnGuardar_Click`. Se activa al hacer clic. Abre la conexión a la base de datos, prepara y ejecuta una consulta SQL `INSERT` para guardar el contacto, cierra la conexión y actualiza el `ListBox`.
        - **Utilización:** Disparador para guardar el contacto en la base de datos.
    - **ListBox:**
        - **Nombre (Name):** `lstContactos`. Muestra los contactos agregados, similar a los programas de agenda anteriores.
        - **Colección `Items`:** Almacena las cadenas que representan cada contacto.
        - **Método `Items.Add()`:** Añade el contacto a la lista visible después de ser guardado en la base de datos.
        - **Utilización:** Presenta la lista de contactos al usuario.
- **Importación de Namespace:**
    - **`Imports System.Data.OleDb`:** Es esencial para trabajar con bases de datos que utilizan la tecnología OLE DB, como Microsoft Access.
- **Objetos y Métodos de Conectividad a Bases de Datos:**
    - **`OleDbConnection` (`conexion`):** Objeto a nivel de clase que representa la conexión a la base de datos. Se inicializa con una cadena de conexión que especifica el proveedor (`Provider=Microsoft.ACE.OLEDB.12.0;`) y la ruta del archivo de la base de datos (`Data Source=agenda.accdb;`).
    - **`OleDbCommand` (`cmd`):** Objeto que representa un comando SQL a ejecutar en la base de datos. Se crea con la consulta `INSERT` para añadir un nuevo contacto a la tabla `Contactos`. Se utiliza una consulta parametrizada (`@n`, `@t`) para mejorar la seguridad y prevenir la inyección SQL.
    - **`cmd.Parameters.AddWithValue("@n", txtNombre.Text)`:** Este método añade un parámetro a la consulta SQL. `@n` es el nombre del parámetro en la consulta, y `txtNombre.Text` es el valor que se le asigna.
    - **`conexion.Open()`:** Establece la conexión con la base de datos.
    - **`cmd.ExecuteNonQuery()`:** Ejecuta la consulta SQL. Se usa para comandos que no devuelven datos (como `INSERT`, `UPDATE`, `DELETE`), y retorna el número de filas afectadas.

    - **`conexion.Close()`:** Cierra la conexión a la base de datos, liberando recursos.
- **Requisito Adicional:** Se necesita una base de datos Microsoft Access llamada `agenda.accdb` con una tabla `Contactos` que contenga al menos dos campos: `Nombre` y `Telefono`.

Programa 81. Formulario con gráficos simples

- **Explicación del Ejercicio:** Este programa demuestra cómo **dibujar formas geométricas básicas** (líneas, rectángulos, elipses) directamente sobre un control `PictureBox` en tiempo de ejecución. Utiliza la clase `Graphics` de .NET para realizar estas operaciones de dibujo.
- **Controles Necesarios:**
    - **Button:**
        - **Nombre (Name):** `btnDibujar`.
        - **Propiedad `Text`:** "Dibujar". Texto visible en el botón.
        - **Evento `Click`:** `btnDibujar_Click`. Se activa al hacer clic. Este evento obtiene un objeto `Graphics` para el `PictureBox`, limpia su superficie de dibujo, y luego invoca los métodos de dibujo para trazar las diferentes formas.
        - **Utilización:** Inicia el proceso de dibujo en el `PictureBox`.
    - **PictureBox:**
        - **Nombre (Name):** `picGrafico`. Es el lienzo donde se realizarán los dibujos.
        - **Propiedad `Size`:** 300x200. Establece las dimensiones del `PictureBox` en el formulario.
        - **Propiedad `BorderStyle`:** `FixedSingle`. Añade un borde visible de una sola línea al `PictureBox`, lo que ayuda a delimitar el área de dibujo.
        - **Utilización:** Proporciona un área visual para dibujar gráficos.
- **Clases y Métodos de Dibujo:**
    - **`Graphics` (`g`):** Un objeto de esta clase representa la superficie de dibujo (en este caso, la superficie del `PictureBox`). Se obtiene utilizando `picGrafico.CreateGraphics()`.
    - **`g.Clear(Color.White)`:** Este método limpia toda la superficie de dibujo (`PictureBox`) y la rellena con el color especificado (blanco en este caso). Esto es útil para eliminar dibujos anteriores antes de crear nuevos.
    - **`Pens`:** Es una clase estática que proporciona acceso a objetos `Pen` predefinidos con diferentes colores y anchos. Un `Pen` se utiliza para dibujar líneas y los bordes de las formas.
        - **`Pens.Blue`**, **`Pens.Red`**, **`Pens.Green`:** Ejemplos de `Pen` de colores específicos utilizados para trazar las formas.
    - **`g.DrawLine(pen, x1, y1, x2, y2)`:** Dibuja una línea entre las coordenadas inicial (`x1`, `y1`) y final (`x2`, `y2`) utilizando el `Pen` especificado.
    - **`g.DrawRectangle(pen, x, y, width, height)`:** Dibuja el contorno de un rectángulo. Los parámetros `x`, `y` definen la esquina superior izquierda, y `width`, `height` definen sus dimensiones.

- **`g.DrawEllipse(pen, x, y, width, height)`:** Dibuja el contorno de una elipse. Los parámetros `x`, `y`, `width`, `height` definen el rectángulo delimitador dentro del cual se inscribe la elipse.

Programa 82. Formulario con impresión de texto

- **Explicación del Ejercicio:** Este programa ilustra cómo integrar la funcionalidad de **impresión** en una aplicación. Permite al usuario tomar el texto de un `TextBox` y enviarlo a una impresora, utilizando los controles `PrintDocument` y `PrintDialog` de .NET para gestionar el proceso de impresión y la interfaz con el usuario.
- **Controles Necesarios:**
  - **TextBox:**
    - **Nombre (Name):** `txtTexto`. Es el área donde el usuario puede escribir o ver el texto que se desea imprimir.
    - **Propiedad `Multiline`:** `True`. Permite la entrada y visualización de múltiples líneas de texto, lo que es útil para contenido a imprimir.
    - **Propiedad `Size`:** 300x150. Define las dimensiones iniciales del `TextBox`.
    - **Propiedad `Text`:** Contiene el texto que será impreso.
    - **Utilización:** Sirve como el origen del contenido para la impresión.
  - **Button:**
    - **Nombre (Name):** `btnImprimir`.
    - **Propiedad `Text`:** "Imprimir". Texto visible en el botón.
    - **Evento `Click`:** `btnImprimir_Click`. Se activa al hacer clic. Muestra el `PrintDialog1` al usuario. Si el usuario confirma la impresión (`DialogResult.OK`), llama al método `PrintDocument1.Print()` para iniciar el proceso de impresión.
    - **Utilización:** Es el disparador para iniciar el cuadro de diálogo de impresión.
  - **PrintDocument:**
    - **Nombre (Name):** `PrintDocument1`. Componente no visible que encapsula la lógica para imprimir un documento.
    - **Método `Print()`:** Inicia el proceso de impresión. Al ser llamado, dispara el evento `PrintPage` para cada página que se va a imprimir.
    - **Utilización:** Gestiona el documento que se va a imprimir.
  - **PrintDialog:**
    - **Nombre (Name):** `PrintDialog1`. Componente no visible que muestra el cuadro de diálogo estándar de Windows para seleccionar una impresora y configurar opciones de impresión.
    - **Método `ShowDialog()`:** Muestra el cuadro de diálogo al usuario. Retorna `DialogResult.OK` si el usuario hace clic en el botón "Imprimir" del diálogo.
    - **Utilización:** Permite al usuario seleccionar la impresora y configurar las opciones de impresión antes de que comience el proceso.
- **Evento de Impresión y Dibujo en la Página:**
  - **`PrintDocument1_PrintPage` (Evento):** Este evento se dispara automáticamente por el `PrintDocument` cada vez que necesita "dibujar" el

contenido de una página para la impresión. Es aquí donde se especifica lo que aparecerá en el papel.

- **`e As Printing.PrintPageEventArgs`:** El objeto `e` de tipo `PrintPageEventArgs` proporciona acceso al objeto `Graphics` de la página (`e.Graphics`) y a otras propiedades de la página (márgenes, tamaño).
- **`e.Graphics.DrawString(txtTexto.Text, New Font("Arial", 12), Brushes.Black, 50, 50)`:** Este método dibuja el texto de `txtTexto.Text` en la página de impresión. Se especifica una nueva fuente (`Arial`, tamaño 12), un color de pincel (`Brushes.Black`) y las coordenadas (`x=50`, `y=50`) donde comenzará el texto en la página.

Programa 83. Calculadora de ecuaciones de segundo grado

- **Explicación del Ejercicio:** Este programa es una **calculadora de ecuaciones de segundo grado** de la forma $Ax^2 + Bx + C = 0$. El usuario introduce los coeficientes A, B y C en cuadros de texto, y al hacer clic en un botón, el programa calcula y muestra las soluciones reales (raíces) de la ecuación. También informa si no existen soluciones reales.
- **Controles Necesarios:**
  - **TextBox:**
    - **Nombre (Name):** `txtA`. Campo de entrada para el coeficiente 'A'.
    - **Propiedad `Text`:** Contiene el valor del coeficiente A. Se convierte a `Double` utilizando `Val()` para los cálculos.
    - **Utilización:** Permite ingresar el coeficiente cuadrático.
  - **TextBox:**
    - **Nombre (Name):** `txtB`. Campo de entrada para el coeficiente 'B'.
    - **Propiedad `Text`:** Contiene el valor del coeficiente B. Se convierte a `Double` con `Val()`.
    - **Utilización:** Permite ingresar el coeficiente lineal.
  - **TextBox:**
    - **Nombre (Name):** `txtC`. Campo de entrada para el coeficiente 'C'.
    - **Propiedad `Text`:** Contiene el valor del coeficiente C. Se convierte a `Double` con `Val()`.
    - **Utilización:** Permite ingresar el término independiente.
  - **Button:**
    - **Nombre (Name):** `btnCalcular`.
    - **Propiedad `Text`:** "Calcular". Texto visible en el botón.
    - **Evento `Click`:** `btnCalcular_Click`. Se activa al hacer clic. Obtiene los coeficientes, calcula el discriminante, determina si hay soluciones reales y las calcula/muestra.
    - **Utilización:** Inicia la resolución de la ecuación.
  - **Label:**
    - **Nombre (Name):** `lblResultado`.

        - **Propiedad `Text`:** Muestra las soluciones `x1` y `x2` (ej., "x1=2, x2=-3") o el mensaje "No hay soluciones reales" si el discriminante es negativo.
        - **Utilización:** Presenta los resultados de la ecuación al usuario.
- **Fórmulas y Lógica Matemática:**
    - **Discriminante (`d`):** `d = b ^ 2 - 4 * a * c`. Este valor determina la naturaleza de las raíces:
        - Si `d < 0`: No hay soluciones reales.
        - Si `d >= 0`: Hay una o dos soluciones reales.
    - **Fórmula General (Raíces):**
        - `x1 = (-b + Math.Sqrt(d)) / (2 * a)`
        - `x2 = (-b - Math.Sqrt(d)) / (2 * a)`
    - **`Math.Sqrt(d)`:** Método de la clase `Math` que calcula la raíz cuadrada del discriminante `d`.

Programa 84. Calculadora de interés simple

- **Explicación del Ejercicio:** Este programa calcula el **interés simple** generado por una inversión o un préstamo. El usuario introduce el capital inicial, la tasa de interés anual (en porcentaje) y el tiempo en años. Al hacer clic en un botón, el programa muestra el monto del interés simple calculado.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtCapital`. Campo de entrada para el monto principal (capital).
        - **Propiedad `Text`:** Contiene el valor numérico del capital. Se convierte a `Double` usando `Val()` para el cálculo.
        - **Utilización:** Permite ingresar el capital inicial.
    - **TextBox:**
        - **Nombre (Name):** `txtInteres`. Campo de entrada para la tasa de interés anual (ej., "5" para 5%).
        - **Propiedad `Text`:** Contiene el valor numérico de la tasa. Se convierte a `Double` con `Val()` y luego se divide por 100 para obtener la tasa en formato decimal (ej., 0.05).
        - **Utilización:** Permite ingresar la tasa de interés.
    - **TextBox:**
        - **Nombre (Name):** `txtTiempo`. Campo de entrada para el período de tiempo en años.
        - **Propiedad `Text`:** Contiene el valor numérico del tiempo. Se convierte a `Double` con `Val()`.
        - **Utilización:** Permite ingresar el tiempo de la inversión/préstamo.
    - **Button:**
        - **Nombre (Name):** `btnCalcular`.
        - **Propiedad `Text`:** "Calcular". Texto visible en el botón.

        - **Evento `Click`:** `btnCalcular_Click`. Se activa al hacer clic. Obtiene los valores de capital, tasa de interés y tiempo, aplica la fórmula del interés simple y muestra el resultado en la etiqueta.
        - **Utilización:** Inicia el cálculo del interés.
    - **Label:**
        - **Nombre (Name):** `lblResultado`.
        - **Propiedad `Text`:** Muestra el interés simple calculado (ej., "Interés simple: 100.00").
        - **Utilización:** Presenta el resultado del cálculo al usuario.
- **Fórmula del Interés Simple:**
    - `interés = C * i * t` (donde C es Capital, i es tasa de interés anual en decimal, y t es tiempo en años).

Programa 85. Calculadora de interés compuesto

- **Explicación del Ejercicio:** Este programa calcula el **monto final de una inversión con interés compuesto**. El usuario introduce el capital inicial, la tasa de interés anual (en porcentaje) y el tiempo en años. A diferencia del interés simple, el interés compuesto calcula el interés sobre el capital inicial más los intereses acumulados de períodos anteriores.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtCapital`. Campo de entrada para el monto principal (capital).
        - **Propiedad `Text`:** Contiene el valor numérico del capital. Se convierte a `Double` usando `Val()` para el cálculo.
        - **Utilización:** Permite ingresar el capital inicial.
    - **TextBox:**
        - **Nombre (Name):** `txtInteres`. Campo de entrada para la tasa de interés anual (en porcentaje).
        - **Propiedad `Text`:** Contiene el valor numérico de la tasa. Se convierte a `Double` con `Val()` y luego se divide por 100 para obtener la tasa en formato decimal.
        - **Utilización:** Permite ingresar la tasa de interés.
    - **TextBox:**
        - **Nombre (Name):** `txtTiempo`. Campo de entrada para el período de tiempo en años.
        - **Propiedad `Text`:** Contiene el valor numérico del tiempo. Se convierte a `Double` con `Val()`.
        - **Utilización:** Permite ingresar el tiempo de la inversión.
    - **Button:**
        - **Nombre (Name):** `btnCalcular`.
        - **Propiedad `Text`:** "Calcular". Texto visible en el botón.
        - **Evento `Click`:** `btnCalcular_Click`. Se activa al hacer clic. Obtiene los valores de capital, tasa de interés y tiempo, aplica la fórmula del interés compuesto y muestra el resultado en la etiqueta.

        - **Utilización:** Inicia el cálculo del monto.
    - **Label:**
        - **Nombre (Name):** `lblResultado`.
        - **Propiedad `Text`:** Muestra el monto final con interés compuesto, formateado a dos decimales (`monto.ToString("0.00")`).
        - **Utilización:** Presenta el resultado del cálculo al usuario.
- **Fórmula del Interés Compuesto:**
    - `monto = C * (1 + i) ^ t` (donde C es Capital, i es tasa de interés anual en decimal, y t es tiempo en años).

Programa 86. Conversor de números decimales a binarios

- **Explicación del Ejercicio:** Este programa convierte un número entero **decimal (base 10)** ingresado por el usuario a su representación **binaria (base 2)**. Utiliza un método integrado de .NET para realizar la conversión de manera eficiente.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtDecimal`. Campo de entrada para el número decimal.
        - **Propiedad `Text`:** Contiene el valor numérico decimal que el usuario desea convertir. Se convierte a `Integer` usando `Val()` antes de la conversión a binario.
        - **Utilización:** Permite al usuario introducir el número decimal.
    - **Button:**
        - **Nombre (Name):** `btnConvertir`.
        - **Propiedad `Text`:** "Convertir". Texto visible en el botón.
        - **Evento `Click`:** `btnConvertir_Click`. Se activa al hacer clic. Obtiene el número decimal, realiza la conversión a binario y muestra el resultado en la etiqueta.
        - **Utilización:** Inicia el proceso de conversión.
    - **Label:**
        - **Nombre (Name):** `lblBinario`.
        - **Propiedad `Text`:** Muestra la representación binaria del número decimal introducido (ej., "Binario: 11001").
        - **Utilización:** Presenta el resultado de la conversión al usuario.
- **Método de Conversión:**
    - **`Convert.ToString(n, 2)`:** Este método estático de la clase `Convert` es la clave de este programa. Toma un número entero (`n`) y lo convierte a una representación de cadena en la base numérica especificada (en este caso, `2` para binario).

Programa 87. Conversor de binarios a decimales

- **Explicación del Ejercicio:** Este programa realiza la operación inversa al anterior: **convierte un número binario (base 2)** ingresado como texto por el usuario a su equivalente **decimal (base 10)**. Incluye un bloque `Try-Catch` para manejar posibles errores si el usuario introduce una cadena que no es un número binario válido.

- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtBinario`. Campo de entrada para el número binario en formato de cadena.
        - **Propiedad `Text`:** Contiene la cadena de dígitos binarios (solo '0' y '1') que el usuario desea convertir.
        - **Utilización:** Permite al usuario introducir el número binario.
    - **Button:**
        - **Nombre (Name):** `btnConvertir`.
        - **Propiedad `Text`:** "Convertir". Texto visible en el botón.
        - **Evento `Click`:** `btnConvertir_Click`. Se activa al hacer clic. Obtiene la cadena binaria, intenta convertirla a decimal y muestra el resultado o un mensaje de error.
        - **Utilización:** Inicia el proceso de conversión.
    - **Label:**
        - **Nombre (Name):** `lblDecimal`.
        - **Propiedad `Text`:** Muestra el número decimal resultante (ej., "Decimal: 25") o el mensaje "Número binario inválido" en caso de error de formato.
        - **Utilización:** Presenta el resultado de la conversión al usuario.
- **Método de Conversión y Manejo de Errores:**
    - **`Convert.ToInt32(bin, 2)`:** Este método estático de la clase `Convert` es fundamental. Toma una cadena (`bin`) que representa un número en una base específica (en este caso, `2` para binario) y la convierte a un número entero decimal de 32 bits.
    - **`Try...Catch ex As Exception`:** Este bloque es crucial para la robustez del programa. Si el texto introducido en `txtBinario` no es una cadena binaria válida (contiene caracteres que no son '0' o '1'), el método `Convert.ToInt32()` lanzará una excepción. El bloque `Catch` intercepta esta excepción y permite al programa mostrar un mensaje de error amigable en `lblDecimal.Text` en lugar de fallar.

Programa 88. Conversor de números romanos a enteros

- **Explicación del Ejercicio:** Este programa convierte un número romano ingresado por el usuario a su equivalente **entero decimal**. Implementa la lógica para interpretar los símbolos romanos y sus combinaciones (como IV o IX) y calcula el valor total.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtRomano`. Campo de entrada para el número romano (ej., "XIV").
        - **Propiedad `Text`:** Contiene la cadena del número romano. Se convierte a mayúsculas con `ToUpper()` antes de ser procesada para asegurar la uniformidad en la comparación de símbolos.
        - **Utilización:** Permite al usuario introducir el número romano.
    - **Button:**
        - **Nombre (Name):** `btnConvertir`.

        - **Propiedad `Text`:** "Convertir". Texto visible en el botón.
        - **Evento `Click`:** `btnConvertir_Click`. Se activa al hacer clic. Contiene la lógica principal del algoritmo de conversión de romano a entero.
        - **Utilización:** Inicia el proceso de conversión.
    - **Label:**
        - **Nombre (Name):** `lblEntero`.
        - **Propiedad `Text`:** Muestra el número entero decimal resultante (ej., "Entero: 14").
        - **Utilización:** Presenta el resultado de la conversión al usuario.
- **Variables y Estructuras de Datos de Soporte (No controles visuales):**
    - **`Dim dic As New Dictionary(Of Char, Integer)`:** Se utiliza un `Dictionary` para almacenar el valor numérico asociado a cada símbolo romano (ej., 'I' es 1, 'V' es 5, 'X' es 10, etc.). Esto permite una búsqueda rápida del valor de cada carácter romano.
    - **`total` (Integer):** Acumula el valor entero resultante a medida que se procesa el número romano.
    - **`prev` (Integer):** Almacena el valor del símbolo romano anterior procesado. Es crucial para manejar la lógica de resta (ej., en IV, el valor de 'V' (5) es mayor que 'I' (1), por lo que se resta 1 a 5).
    - **Bucle `For i As Integer = roman.Length - 1 To 0 Step -1`:** El algoritmo itera sobre la cadena romana de derecha a izquierda (desde el último carácter hasta el primero).
    - **Lógica de Conversión:** Dentro del bucle, se obtiene el valor del símbolo actual (`val`). Se compara `val` con `prev`. Si `val < prev`, se resta `val` al `total` (regla de resta, ej. "IV" -> 5 - 1 = 4). De lo contrario, se suma `val` al `total`. Luego, `prev` se actualiza con `val` para la siguiente iteración.

Programa 89. Conversor de horas a minutos y segundos

- **Explicación del Ejercicio:** Este programa convierte una cantidad de tiempo ingresada en **horas** a sus equivalentes en **minutos y segundos**. Es una utilidad sencilla que muestra todos los resultados de la conversión en una única etiqueta.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtHoras`. Campo de entrada para la cantidad de horas.
        - **Propiedad `Text`:** Contiene el valor numérico en horas. Se convierte a `Double` utilizando `Val()` para los cálculos.
        - **Utilización:** Permite al usuario introducir la cantidad de horas.
    - **Button:**
        - **Nombre (Name):** `btnConvertir`.
        - **Propiedad `Text`:** "Convertir". Texto visible en el botón.
        - **Evento `Click`:** `btnConvertir_Click`. Se activa al hacer clic. Obtiene el valor en horas, calcula los minutos y segundos correspondientes y actualiza la etiqueta de resultado.
        - **Utilización:** Inicia el proceso de conversión de tiempo.

        - **Label:**
            - **Nombre (Name):** `lblResultado`.
            - **Propiedad `Text`:** Muestra la cantidad original en horas y sus equivalentes en minutos y segundos en un formato legible (ej., "2 horas = 120 minutos = 7200 segundos").
            - **Utilización:** Presenta el resultado de la conversión al usuario.
    - **Fórmulas de Conversión:**
        - Minutos: `h * 60` (donde `h` es el número de horas).
        - Segundos: `h * 3600` (ya que 1 hora = 60 minutos * 60 segundos = 3600 segundos).

Programa 90. Conversor de kilogramos a libras

- **Explicación del Ejercicio:** Este programa es una utilidad sencilla para **convertir una cantidad de peso de kilogramos (kg) a libras (lb)**. El usuario introduce la cantidad en kilogramos, y al presionar un botón, el programa muestra el equivalente en libras, formateado a dos decimales.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtKg`. Campo de entrada para el peso en kilogramos.
        - **Propiedad `Text`:** Contiene el valor numérico en kilogramos. Se convierte a `Double` utilizando `Val()` para el cálculo.
        - **Utilización:** Permite al usuario introducir el peso en kilogramos.
    - **Button:**
        - **Nombre (Name):** `btnConvertir`.
        - **Propiedad `Text`:** "Convertir". Texto visible en el botón.
        - **Evento `Click`:** `btnConvertir_Click`. Se activa al hacer clic. Obtiene la cantidad en kilogramos, realiza la conversión a libras y actualiza la etiqueta de resultado.
        - **Utilización:** Inicia el proceso de conversión de peso.
    - **Label:**
        - **Nombre (Name):** `lblResultado`.
        - **Propiedad `Text`:** Muestra la cantidad original en kilogramos y su equivalente en libras, formateado a dos decimales con `lb.ToString("0.00")` (ej., "10 kg = 22.05 lb").
        - **Utilización:** Presenta el resultado de la conversión al usuario.
- **Fórmula de Conversión:**
    - Libras: `kg * 2.20462` (donde `kg` es el número de kilogramos). El factor `2.20462` es la constante de conversión de kilogramos a libras.

Programa 91. Conversor de metros a pies

- **Explicación del Ejercicio:** Este programa permite al usuario **convertir una longitud dada en metros a su equivalente en pies**. El usuario introduce la cantidad en metros en un `TextBox`, y el resultado se muestra en un `Label` al hacer clic en un botón.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtMetros`.
        - **Propiedad `Text`:** Contiene el valor numérico en metros que el usuario ingresa. Se convierte a tipo `Double` utilizando `Val()` para el cálculo.
        - **Utilización:** Es el campo de entrada donde el usuario introduce la longitud en metros que desea convertir.
    - **Button:**
        - **Nombre (Name):** `btnConvertir`.
        - **Propiedad `Text`:** "Convertir". Texto visible en el botón que indica su función.
        - **Evento `Click`:** `btnConvertir_Click`. Se activa cuando el usuario pulsa el botón. Dentro de este evento, se recupera el valor de los metros, se realiza la conversión a pies y se actualiza el `lblResultado`.
        - **Utilización:** Sirve como el disparador para iniciar el proceso de conversión.
    - **Label:**
        - **Nombre (Name):** `lblResultado`.
        - **Propiedad `Text`:** Muestra la longitud original en metros y su equivalente en pies. El valor de los pies se formatea a dos decimales con `ToString("0.00")` para una presentación clara.
        - **Utilización:** Presenta el resultado de la conversión al usuario.
- **Explicación de la Lógica:**
    - En el evento `btnConvertir_Click`, se obtiene el valor de `txtMetros.Text` y se almacena en la variable `metros` de tipo `Double`.
    - La conversión se realiza multiplicando `metros` por el factor de conversión `3.28084` (que es el número de pies en un metro) y el resultado se guarda en la variable `pies`.
    - El `lblResultado.Text` se actualiza para mostrar `metros` en metros y `pies` en pies, formateando los pies a dos decimales.

Programa 92. Conversor de Celsius a Kelvin

- **Explicación del Ejercicio:** Este programa convierte una temperatura dada en **grados Celsius a su equivalente en Kelvin**. El usuario introduce la temperatura en Celsius en un `TextBox`, y el resultado se muestra en un `Label` al hacer clic en un botón.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtCelsius`.
        - **Propiedad `Text`:** Contiene el valor numérico de la temperatura en Celsius que el usuario ingresa. Se convierte a `Double` utilizando `Val()` para el cálculo.

        - **Utilización:** Es el campo donde el usuario introduce la temperatura en Celsius a convertir.
    - **Button:**
        - **Nombre (Name):** `btnConvertir`.
        - **Propiedad `Text`:** "Convertir".
        - **Evento `Click`:** `btnConvertir_Click`. Se ejecuta al hacer clic en el botón. Contiene la lógica para obtener la temperatura en Celsius, realizar la conversión a Kelvin y mostrar el resultado.
        - **Utilización:** Inicia el proceso de conversión de temperatura.
    - **Label:**
        - **Nombre (Name):** `lblResultado`.
        - **Propiedad `Text`:** Presenta la temperatura original en Celsius y la convertida en Kelvin. El valor en Kelvin se formatea a dos decimales (`ToString("0.00")`) para mayor precisión y legibilidad.
        - **Utilización:** Muestra el resultado de la conversión al usuario.
- **Explicación de la Lógica:**
    - Al hacer clic en `btnConvertir`, el valor de `txtCelsius.Text` se extrae y se convierte a `Double` en la variable `c`.
    - La conversión a Kelvin se realiza sumando `273.15` al valor en Celsius (`c`), y el resultado se guarda en la variable `k`.
    - El `lblResultado.Text` se actualiza para mostrar ambas temperaturas en un formato legible, formateando el valor Kelvin a dos decimales.

Programa 93. Conversor de binario a hexadecimal

- **Explicación del Ejercicio:** Este programa convierte un número **binario (base 2)** ingresado por el usuario a su representación **hexadecimal (base 16)**. Incluye manejo de errores para números binarios inválidos.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtBinario`.
        - **Propiedad `Text`:** Contiene la cadena de dígitos binarios (solo '0' y '1') que el usuario desea convertir.
        - **Utilización:** Permite al usuario introducir el número binario.
    - **Button:**
        - **Nombre (Name):** `btnConvertir`.
        - **Propiedad `Text`:** "Convertir".
        - **Evento `Click`:** `btnConvertir_Click`. Gestiona la lógica de conversión, incluyendo el manejo de errores.
        - **Utilización:** Inicia el proceso de conversión de binario a hexadecimal.
    - **Label:**
        - **Nombre (Name):** `lblHex`.
        - **Propiedad `Text`:** Muestra la representación hexadecimal del número binario (ej., "Hexadecimal: 19") o un mensaje de error si el binario es inválido.

        - **Utilización:** Presenta el resultado de la conversión o mensajes de error al usuario.
- **Explicación de la Lógica:**
    - El evento `btnConvertir_Click` está envuelto en un bloque `Try-Catch` para manejar excepciones que puedan ocurrir si el `txtBinario.Text` no es un formato binario válido.
    - Dentro del `Try`, `Convert.ToInt32(txtBinario.Text, 2)` es la clave. Convierte la cadena binaria a un número entero decimal, especificando la base 2 como origen.
    - Luego, la función `Hex(n)` convierte ese número entero decimal a su representación hexadecimal, que se asigna a `lblHex.Text`.
    - Si se produce una excepción (por un binario inválido), el bloque `Catch` asigna el mensaje "Número binario inválido" a `lblHex.Text`.

Programa 94. Conversor de hexadecimal a decimal

- **Explicación del Ejercicio:** Este programa realiza la operación inversa al anterior: convierte un número **hexadecimal (base 16)** ingresado por el usuario a su equivalente **decimal (base 10)**. Al igual que el programa 93, incluye manejo de errores para entradas inválidas.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtHex`.
        - **Propiedad `Text`:** Contiene la cadena de dígitos hexadecimales (0-9, A-F) que el usuario desea convertir.
        - **Utilización:** Permite al usuario introducir el número hexadecimal.
    - **Button:**
        - **Nombre (Name):** `btnConvertir`.
        - **Propiedad `Text`:** "Convertir".
        - **Evento `Click`:** `btnConvertir_Click`. Contiene la lógica de conversión y el manejo de errores.
        - **Utilización:** Inicia el proceso de conversión de hexadecimal a decimal.
    - **Label:**
        - **Nombre (Name):** `lblDecimal`.
        - **Propiedad `Text`:** Muestra el número decimal resultante (ej., "Decimal: 25") o el mensaje "Número hexadecimal inválido" en caso de un formato incorrecto.
        - **Utilización:** Presenta el resultado de la conversión o mensajes de error al usuario.
- **Explicación de la Lógica:**
    - El evento `btnConvertir_Click` utiliza un bloque `Try-Catch` para gestionar errores de formato en la entrada hexadecimal.
    - Dentro del `Try`, `Convert.ToInt32(txtHex.Text, 16)` es el método clave. Toma la cadena hexadecimal de `txtHex.Text` y la convierte a un número entero decimal, indicando que la base de origen es 16.
    - El número decimal resultante se asigna a `lblDecimal.Text`.

- Si el formato de la cadena es incorrecto, el bloque `Catch` intercepta la excepción y muestra el mensaje "Número hexadecimal inválido" en `lblDecimal.Text`.

Programa 95. Juego de memoria con botones

- **Explicación del Ejercicio:** Este programa implementa un **juego de memoria simple** con cuatro botones que cambian de color. El usuario debe recordar y reproducir una secuencia predefinida de colores.
- **Controles Necesarios:**
  - **Button:**
    - **Nombres (Names):** `btn1`, `btn2`, `btn3`, `btn4`. Se necesitan cuatro botones para este juego.
    - **Propiedad `BackColor`:** Representa el color de fondo del botón. Los botones cambiarán su color al hacer clic y se comparará este color con la secuencia esperada.
    - **Evento `Click`:** `btn_Click`. Este evento se maneja para todos los botones de forma compartida (`Handles btn1.Click, btn2.Click, ...`). Contiene la lógica del juego para verificar si el color del botón pulsado coincide con el de la secuencia.
    - **Utilización:** Los botones son los elementos interactivos que el usuario presiona para intentar reproducir la secuencia de colores.
- **Variables y Clases de Soporte (No controles visuales):**
  - **`colores()` (Array de Color):** Almacena una lista de colores predefinidos (Rojo, Verde, Azul, Amarillo) que los botones pueden mostrar.
  - **`secuencia()` (Array de Integer):** Define la secuencia esperada de colores, utilizando los índices del array `colores`. Por ejemplo, `{0, 1, 2, 3}` podría significar Rojo, Verde, Azul, Amarillo.
  - **`indice` (Integer):** Variable que lleva la cuenta de la posición actual en la `secuencia` que el jugador debe adivinar.
- **Explicación de la Lógica:**
  - Cuando se hace clic en cualquier botón, el evento `btn_Click` se ejecuta.
  - `CType(sender, Button)` convierte el objeto `sender` (que es el botón que se hizo clic) a su tipo `Button`.
  - Se compara el `BackColor` del botón clicado (`btn.BackColor`) con el color esperado en la `secuencia` actual (`colores(secuencia(indice))`).
  - Si la elección del color es correcta, la variable `indice` se incrementa. Si `indice` alcanza la longitud total de la `secuencia`, el jugador ha ganado, se muestra un `MessageBox` y `indice` se reinicia a 0.
  - Si la elección del color es incorrecta, se muestra un `MessageBox` de "Error" y `indice` también se reinicia.
  - **Nota:** Para que el juego sea más dinámico, la `secuencia` y/o los `BackColor` iniciales de los botones deberían generarse aleatoriamente al inicio del juego, lo cual no está explícitamente en el código proporcionado pero es una extensión lógica.

Programa 96. Juego de laberinto simple

- **Explicación del Ejercicio:** Este programa simula un **juego de laberinto simple** donde el usuario controla un "jugador" (representado por un botón) utilizando las teclas de dirección. El objetivo es alcanzar una "meta" (otro botón).
- **Controles Necesarios:**
    - **Panel:**
        - **Nombre (Name):** `pnlLaberinto`. Aunque el código proporcionado no lo utiliza para dibujar las paredes, se sugiere su uso para contener visualmente el laberinto.
        - **Utilización:** Actúa como el contenedor del área de juego y de los elementos del laberinto.
    - **Button (Jugador):**
        - **Nombre (Name):** `btnJugador`.
        - **Propiedad `Text`:** "J". Identifica visualmente al jugador.
        - **Propiedad `Top`:** Representa la coordenada Y (posición vertical) de la parte superior del botón. Se modifica para mover el jugador hacia arriba y abajo.
        - **Propiedad `Left`:** Representa la coordenada X (posición horizontal) del lado izquierdo del botón. Se modifica para mover el jugador hacia la izquierda y derecha.
        - **Propiedad `Bounds`:** Define el tamaño y la ubicación del control. Se utiliza para detectar si se superpone con la meta (`btnMeta.Bounds`).
        - **Utilización:** Es el elemento interactivo que el usuario mueve a través del laberinto.
    - **Button (Meta):**
        - **Nombre (Name):):** `btnMeta`.
        - **Propiedad `Text`:** "M". Identifica visualmente la meta.
        - **Propiedad `Bounds`:** Se utiliza para determinar la posición y el tamaño de la meta para la detección de colisiones.
        - **Utilización:** Representa el objetivo que el jugador debe alcanzar en el laberinto.
- **Configuración Adicional (del Formulario):**
    - **Propiedad `KeyPreview` (del Formulario):** Debe establecerse en `True`. Esta propiedad es fundamental porque permite al formulario capturar los eventos de teclado (como `KeyDown`) antes que cualquier control individual del formulario. Esto asegura que el movimiento del jugador funcione correctamente.
- **Explicación de la Lógica:**
    - El evento `Form1_KeyDown` (o `btnJugador_KeyDown` como referencia en la fuente, pero el código usa `Me.KeyDown` que es el evento del formulario) se dispara cada vez que el usuario presiona una tecla mientras el formulario tiene el foco (gracias a `KeyPreview = True`).
    - Un bloque `Select Case e.KeyCode` evalúa qué tecla de dirección (`Keys.Up`, `Keys.Down`, `Keys.Left`, `Keys.Right`) fue presionada.
    - Dependiendo de la tecla, las propiedades `Top` o `Left` de `btnJugador` se ajustan en `5` píxeles, moviendo el botón del jugador en la dirección correspondiente.

- Después de cada movimiento, `btnJugador.Bounds.IntersectsWith(btnMeta.Bounds)` verifica si los límites del `btnJugador` se superponen con los límites del `btnMeta`.
- Si hay una superposición (es decir, el jugador ha alcanzado la meta), se muestra un `MessageBox` con el mensaje "¡Has ganado!".

Programa 97. Juego de dados con imágenes

- **Explicación del Ejercicio:** Este programa simula el **lanzamiento de dos dados** y muestra los resultados de forma visual utilizando imágenes de dados en dos controles `PictureBox`.
- **Controles Necesarios:**
  - **PictureBox:**
    - **Nombres (Names):** `picDado1`, `picDado2`.
    - **Propiedad `Image`:** Es la propiedad clave. Se le asigna un objeto `Image` (cargado desde los recursos del proyecto) para mostrar la cara del dado correspondiente al número aleatorio generado.
    - **Utilización:** Son los lienzos visuales donde se exhiben las imágenes de los dados para mostrar los resultados del lanzamiento.
  - **Button:**
    - **Nombre (Name):** `btnTirar`.
    - **Propiedad `Text`:** "Tirar Dados".
    - **Evento `Click`:** `btnTirar_Click`. Se activa cuando el usuario hace clic. Contiene la lógica para generar los números aleatorios de los dados y cargar las imágenes correspondientes.
    - **Utilización:** Inicia la acción de "tirar" los dados.
- **Recursos Adicionales:**
  - Se requiere tener imágenes para cada cara del dado (ej., `dado1.png`, `dado2.png`, hasta `dado6.png`) agregadas a los **recursos del proyecto** (en Visual Studio: `My Project > Resources`). Esto permite acceder a ellas programáticamente mediante `My.Resources.ResourceManager.GetObject()`.
- **Variables y Clases de Soporte (No controles visuales):**
  - **`Random` (`rnd`):** Una instancia de esta clase, declarada a nivel de clase, se utiliza para generar números pseudoaleatorios, simulando el lanzamiento de los dados (`rnd.Next(1, 7)`).
- **Explicación de la Lógica:**
  - Cuando se hace clic en `btnTirar`, se generan dos números aleatorios, `d1` y `d2`, utilizando `rnd.Next(1, 7)`. Este método genera un número entero entre 1 (inclusive) y 7 (exclusivo), resultando en valores de 1 a 6, que representan las caras de un dado.
  - `My.Resources.ResourceManager.GetObject("dado" & d1)` construye dinámicamente el nombre del recurso de imagen (por ejemplo, si `d1` es 3, buscará "dado3") y carga la imagen correspondiente desde los recursos del proyecto.
  - Estas imágenes se asignan a las propiedades `Image` de `picDado1` y `picDado2`, lo que hace que los resultados visuales se muestren al usuario.

Programa 98. Juego de ahorcado

- **Explicación del Ejercicio:** Este programa implementa una **versión simple del juego del ahorcado**. La aplicación tiene una palabra secreta predefinida, y el usuario intenta adivinarla letra por letra. El usuario tiene un número limitado de intentos.
- **Controles Necesarios:**
    - **Label (Palabra):**
        - **Nombre (Name):** `lblPalabra`.
        - **Propiedad `Text`:** Inicialmente muestra la palabra oculta con guiones bajos (ej., "_____"). Se actualiza con el progreso de la adivinanza, revelando las letras correctas que el usuario ha encontrado.
        - **Utilización:** Muestra visualmente la palabra a adivinar y el progreso del jugador.
    - **TextBox:**
        - **Nombre (Name):** `txtLetra`.
        - **Propiedad `Text`:** Contiene la letra ingresada por el usuario para probar. Su contenido se convierte a mayúsculas con `ToUpper()` para una comparación sin distinción de mayúsculas/minúsculas.
        - **Método `Clear()`:** Se utiliza para vaciar el `TextBox` después de cada intento, dejándolo listo para la siguiente letra.
        - **Utilización:** Permite al usuario ingresar una letra por turno.
    - **Button:**
        - **Nombre (Name):** `btnProbar`.
        - **Propiedad `Text`:** "Probar".
        - **Evento `Click`:** `btnProbar_Click`. Se activa al hacer clic. Contiene la lógica principal del juego: procesa la letra ingresada, verifica si está en la palabra secreta, actualiza `lblPalabra`, gestiona los intentos y comprueba si el juego ha terminado (victoria o derrota).
        - **Utilización:** Dispara la acción de probar una letra.
    - **Label (Intentos):**
        - **Nombre (Name):** `lblIntentos`.
        - **Propiedad `Text`:** Muestra el número de intentos restantes del usuario (ej., "Intentos: 6"). Se actualiza cada vez que el usuario comete un error.
        - **Utilización:** Informa al usuario sobre sus "vidas" restantes.
- **Variables de Soporte (No controles visuales):**
    - **`palabra` (String):** Almacena la palabra secreta a adivinar (ej., "VBNET"). Se declara a nivel de clase.
    - **`adivinada` (String):** Almacena la representación de la palabra con guiones bajos, que se va revelando a medida que el usuario acierta letras (ej., "_____"). Se declara a nivel de clase.
    - **`intentos` (Integer):** Almacena el número de vidas o intentos restantes para el jugador (inicialmente 6).
- **Explicación de la Lógica:**
    - En el evento `Form1_Load`, `lblPalabra` se inicializa con la `adivinada` (guiones bajos), y `lblIntentos` muestra el número inicial de intentos.

- Cuando se hace clic en `btnProbar`, se obtiene la letra ingresada por el usuario desde `txtLetra.Text` y se convierte a mayúscula.
- Se crea un array de caracteres (`nueva`) a partir de la cadena `adivinada`. Luego, un bucle `For` itera sobre la `palabra` secreta: si la letra ingresada coincide con un carácter en `palabra`, ese carácter se revela en la posición correspondiente del array `nueva`.
- Después del bucle, `adivinada` se actualiza con la nueva cadena construida a partir de `nueva`, y `lblPalabra.Text` se refresca para mostrar el progreso.
- Si la `palabra` secreta **no contiene** la `letra` ingresada (`If Not palabra.Contains(letra) Then`), los `intentos` se decrementan y `lblIntentos.Text` se actualiza.
- Finalmente, se verifican las condiciones de fin de juego: si `adivinada` es igual a `palabra` (victoria), se muestra "¡Ganaste!"; si `intentos` llega a 0 (derrota), se muestra "¡Perdiste!". `txtLetra` se limpia para el siguiente intento.

Programa 99. Juego de tic-tac-toe

- **Explicación del Ejercicio:** Este programa implementa una versión básica del **juego de tres en raya (tic-tac-toe)** para dos jugadores. Los jugadores se turnan para colocar sus marcas ('X' o 'O') en un tablero representado por botones.
- **Controles Necesarios:**
  - **Button:**
    - **Nombres (Names):** `btn1` a `btn9`. Se necesitan nueve botones, dispuestos en una cuadrícula de 3x3, para representar el tablero de juego.
    - **Propiedad `Text`:** Inicialmente vacía (""). Se actualiza a "X" o "O" cuando un jugador hace clic en el botón, marcando la casilla.
    - **Evento `Click`:** `btn_Click`. Este evento se maneja de forma compartida para los nueve botones (`Handles btn1.Click, btn2.Click, ...`). Contiene la lógica para alternar los turnos y marcar la casilla seleccionada.
    - **Utilización:** Cada botón representa una casilla en el tablero de tres en raya, que los jugadores pueden seleccionar para realizar su jugada.
- **Variables de Soporte (No controles visuales):**
  - **`turno` (Boolean):** Variable a nivel de clase, inicializada a `True`. Controla quién está jugando: `True` para el jugador 'X' y `False` para el jugador 'O'.
- **Explicación de la Lógica:**
  - El evento `btn_Click` se activa cuando cualquiera de los nueve botones es pulsado.
  - `CType(sender, Button)` convierte el objeto `sender` al tipo `Button` para poder manipular sus propiedades.
  - Se verifica si el `btn.Text` del botón pulsado está vacío (`If btn.Text = "" Then`). Esto asegura que el usuario solo pueda marcar casillas que no han sido jugadas previamente.
  - Si la casilla está vacía, `btn.Text` se actualiza a "X" o "O" basándose en el valor actual de `turno` (`If(turno, "X", "O")`). Si `turno` es `True`, se pone "X"; si es `False`, se pone "O".

- Inmediatamente después de marcar la casilla, `turno = Not turno` invierte el valor de `turno`, preparando el juego para el siguiente jugador.
- **Nota:** El código proporcionado solo maneja los turnos y la marcación de casillas; no incluye la lógica necesaria para determinar un ganador o un empate en el juego.

Programa 100. Juego de cartas simple

- **Explicación del Ejercicio:** Este programa simula un **juego de cartas muy simple**. Permite al usuario "sacar" una carta aleatoria de una baraja predefinida y la muestra visualmente en un control `PictureBox`.
- **Controles Necesarios:**
  - **PictureBox:**
    - **Nombre (Name):** `picCarta`.
    - **Propiedad `Image`:** Es la propiedad clave. Se le asigna un objeto `Image` (cargado desde los recursos del proyecto) para mostrar la imagen de la carta aleatoria seleccionada.
    - **Utilización:** Es el lienzo visual donde se exhibe la imagen de la carta que el usuario ha "sacado".
  - **Button:**
    - **Nombre (Name):** `btnSacar`.
    - **Propiedad `Text`:** "Sacar Carta".
    - **Evento `Click`:** `btnSacar_Click`. Se activa cuando el usuario hace clic. Contiene la lógica para seleccionar una carta aleatoria y cargar su imagen.
    - **Utilización:** Inicia la acción de "sacar" una nueva carta.
- **Recursos Adicionales:**
  - Se requiere tener archivos de imagen para las cartas (ej., `c1.png`, `c2.png`, hasta `c5.png`, o `.jpg`) agregados a los **recursos del proyecto** (en Visual Studio: `My Project > Resources`). Esto permite acceder a ellas programáticamente.
- **Variables y Clases de Soporte (No controles visuales):**
  - **`Random` (`rnd`):** Una instancia de esta clase, declarada a nivel de clase, se utiliza para generar números pseudoaleatorios, lo que permite seleccionar una carta al azar.
  - **`cartas()` (Array de String):** Almacena los nombres de los recursos de imagen de las cartas (ej., `{"c1", "c2", "c3", "c4", "c5"}`). Se declara a nivel de clase.
- **Explicación de la Lógica:**
  - Cuando se hace clic en `btnSacar`, se genera un `index` aleatorio utilizando `rnd.Next(cartas.Length)`. Este índice estará dentro del rango válido del array `cartas`, seleccionando uno de los nombres de archivo de carta al azar.
  - `My.Resources.ResourceManager.GetObject(cartas(index))` recupera la imagen correspondiente del recurso del proyecto utilizando el nombre de la carta aleatoria obtenida (ej., "c3").
  - La imagen recuperada se asigna a `picCarta.Image`, mostrando la carta seleccionada al usuario.

Programa 101. Juego de lotería con números aleatorios

- **Explicación del Ejercicio:** Este programa simula un sorteo de lotería **generando un conjunto de 6 números aleatorios** diferentes. Los números generados se muestran en un control `ListBox`.
- **Controles Necesarios:**
    - **Button:**
        - **Nombre (Name):** `btnGenerar`. Identificador del botón.
        - **Texto (Text):** "Generar Números". Texto visible en el botón que indica su funcionalidad.
        - **Evento `Click`:** `btnGenerar_Click`. Este evento se activa cuando el usuario pulsa el botón. Contiene la lógica para limpiar la lista y generar los números aleatorios.
        - **Utilización:** El botón es el elemento interactivo que el usuario presiona para desencadenar la generación de los números de lotería.
    - **ListBox:**
        - **Nombre (Name):** `lstNumeros`. Identificador del control de lista donde se mostrarán los números de lotería.
        - **Colección `Items`:** Esta colección almacena y visualiza los números aleatorios generados.
        - **Método `Items.Clear()`:** Se llama al principio del evento `btnGenerar_Click` para eliminar cualquier número generado previamente, asegurando que solo se muestre el sorteo actual.
        - **Método `Items.Add()`:** Añade cada número aleatorio generado como un nuevo elemento a la lista.
        - **Utilización:** El `ListBox` es el componente principal para presentar visualmente los números del sorteo al usuario.
- **Variables y Clases de Soporte (No controles visuales):**
    - **`Random` (`rnd`):** Una instancia de la clase `Random` se declara a nivel de clase (`Dim rnd As New Random()`) para generar números pseudoaleatorios.
- **Explicación de la Lógica:**
    - Al hacer clic en `btnGenerar`, el evento `btnGenerar_Click` se ejecuta.
    - `lstNumeros.Items.Clear()` limpia el `ListBox`.
    - Un bucle `For` se ejecuta 6 veces para generar 6 números.
    - En cada iteración, `rnd.Next(1, 50)` genera un **número entero aleatorio entre 1 (inclusive) y 50 (exclusivo)**, es decir, del 1 al 49.
    - `lstNumeros.Items.Add()` añade este número aleatorio a la lista.

Programa 102. Juego de ruleta simple

- **Explicación del Ejercicio:** Este programa simula un giro de ruleta, **generando un número aleatorio entre 0 y 36** (incluidos) y mostrando el resultado.
- **Controles Necesarios:**
    - **Button:**
        - **Nombre (Name):** `btnGirar`. Identificador del botón.
        - **Texto (Text):** "Girar Ruleta". Texto visible en el botón.

        - **Evento `Click`:** `btnGirar_Click`. Se activa cuando el usuario presiona el botón. Contiene la lógica para generar el número aleatorio.
        - **Utilización:** El botón inicia la simulación del giro de la ruleta.
    - **Label:**
        - **Nombre (Name):** `lblResultado`. Identificador de la etiqueta.
        - **Propiedad `Text`:** Muestra el número generado por la ruleta (ej. "Número: 15").
        - **Utilización:** Presenta el número resultante del giro al usuario.
- **Variables y Clases de Soporte (No controles visuales):**
    - **`Random` (`rnd`):** Una instancia de la clase `Random` se declara a nivel de clase para generar el número aleatorio.
- **Explicación de la Lógica:**
    - Cuando se hace clic en `btnGirar`, se ejecuta el evento `btnGirar_Click`.
    - `rnd.Next(0, 37)` genera un **número entero aleatorio entre 0 (inclusive) y 37 (exclusivo)**, lo que resulta en un número del 0 al 36, simulando los números de una ruleta.
    - Este número se asigna a `lblResultado.Text`, precedido por la cadena "Número: ".

Programa 103. Juego de preguntas y respuestas

- **Explicación del Ejercicio:** Este programa presenta al usuario una pregunta predefinida y **verifica si la respuesta ingresada es correcta o incorrecta**.
- **Controles Necesarios:**
    - **Label:**
        - **Nombre (Name):** `lblPregunta`. Identificador de la etiqueta que muestra la pregunta.
        - **Propiedad `Text`:** Se inicializa en el evento `Form1_Load` con el texto de la pregunta almacenada en la variable `pregunta`.
        - **Utilización:** Muestra la pregunta al usuario.
    - **TextBox:**
        - **Nombre (Name):** `txtRespuesta`. Campo de entrada donde el usuario escribe su respuesta.
        - **Propiedad `Text`:** Contiene la respuesta ingresada por el usuario, que se compara con la `respuestaCorrecta`.
        - **Utilización:** Permite al usuario introducir su respuesta a la pregunta.
    - **Button:**
        - **Nombre (Name):** `btnVerificar`. Identificador del botón.
        - **Texto (Text):** "Verificar". Texto visible en el botón.
        - **Evento `Click`:** `btnVerificar_Click`. Se dispara cuando el usuario presiona el botón. Contiene la lógica para comparar la respuesta del usuario con la correcta.
        - **Utilización:** El botón inicia la verificación de la respuesta.
    - **Label:**

        - **Nombre (Name):** `lblResultado`. Identificador de la etiqueta que muestra el resultado de la verificación.
        - **Propiedad `Text`:** Se actualiza para indicar si la respuesta fue "Correcto" o "Incorrecto".
        - **Utilización:** Informa al usuario sobre la validez de su respuesta.
- **Variables de Soporte (No controles visuales):**
    - **`pregunta` (String):** Almacena el texto de la pregunta a mostrar.
    - **`respuestaCorrecta` (String):** Almacena la solución correcta a la pregunta.
- **Explicación de la Lógica:**
    - En el evento `Form1_Load`, `lblPregunta` se inicializa con el contenido de la variable `pregunta`.
    - Cuando se hace clic en `btnVerificar`, el texto de `txtRespuesta` se compara con `respuestaCorrecta`.
    - La comparación se realiza **sin distinguir mayúsculas/minúsculas** utilizando el método `.ToLower()` en ambos textos.
    - `lblResultado.Text` se actualiza para mostrar "Correcto" si las respuestas coinciden, o "Incorrecto" en caso contrario.

Programa 104. Juego de carreras con imágenes

- **Explicación del Ejercicio:** Este programa simula una **carrera entre dos "carros"** representados por controles `PictureBox`. Los carros se mueven hacia la derecha en incrementos aleatorios, y la carrera termina cuando uno de ellos alcanza el borde derecho del formulario.
- **Controles Necesarios:**
    - **PictureBox:**
        - **Nombres (Names):** `picCarro1`, `picCarro2`. Se necesitan dos `PictureBox` para representar los carros en la carrera.
        - **Propiedad `Left`:** Representa la coordenada X (posición horizontal) del lado izquierdo del control. Se incrementa para mover los carros hacia la derecha.
        - **Propiedad `Right`:** Propiedad de solo lectura que devuelve la coordenada X del lado derecho del control. Se utiliza para detectar si el carro ha llegado al final del formulario.
        - **Utilización:** Cada `PictureBox` visualiza uno de los carros que compiten. Se espera que contengan imágenes de carros.
    - **Button:**
        - **Nombre (Name):** `btnCorrer`. Identificador del botón.
        - **Texto (Text):** "Iniciar Carrera". Texto visible en el botón.
        - **Evento `Click`:** `btnCorrer_Click`. Se activa cuando el usuario presiona el botón. Inicia el `Timer1` para comenzar la carrera.
        - **Utilización:** El botón es el disparador para comenzar la carrera.
    - **Timer:**
        - **Nombre (Name):** `Timer1`. Componente no visible que genera eventos en intervalos regulares.

        - **Propiedad `Interval`:** 50. Se configura para disparar su evento `Tick` cada 50 milisegundos, lo que crea un movimiento más fluido y rápido en comparación con un intervalo de 1000 ms.
        - **Método `Start()`:** Inicia el temporizador.
        - **Método `Stop()`:** Detiene el temporizador.
        - **Evento `Tick`:** `Timer1_Tick`. En cada disparo, contiene la lógica para mover los carros y verificar si la carrera ha terminado.
        - **Utilización:** El `Timer` es el motor que controla el movimiento de los carros y la detección del final de la carrera.
- **Variables y Clases de Soporte (No controles visuales):**
    - **`Random` (`rnd`):** Una instancia de la clase `Random` se utiliza para generar la aleatoriedad en el movimiento de los carros.
- **Explicación de la Lógica:**
    - Al hacer clic en `btnCorrer`, el `Timer1` se inicia.
    - Cada 50 milisegundos, el evento `Timer1_Tick` se dispara.
    - `picCarro1.Left` y `picCarro2.Left` se incrementan en una cantidad aleatoria (entre 1 y 9 píxeles) utilizando `rnd.Next(1, 10)`, moviendo los carros hacia la derecha.
    - Se comprueba si `picCarro1.Right` o `picCarro2.Right` han alcanzado o superado el borde derecho del formulario (`Me.Width`).
    - Si un carro alcanza el final, `Timer1.Stop()` detiene el temporizador, y se muestra un `MessageBox` para anunciar al ganador.

Programa 105. Juego de disparos con botones

- **Explicación del Ejercicio:** Este programa es un **juego de disparos conceptual muy simple**. El usuario "dispara" haciendo clic en un botón, y la aplicación simula un "acierto" o "fallo" mediante un `MessageBox`. Lleva un registro del puntaje.
- **Controles Necesarios:**
    - **Button:**
        - **Nombre (Name):** `btnDisparar`. Identificador del botón de disparo.
        - **Texto (Text):** "Disparar". Texto visible en el botón.
        - **Evento `Click`:** `btnDisparar_Click`. Se activa cuando el usuario pulsa el botón. Contiene la lógica para simular el disparo y actualizar el puntaje.
        - **Utilización:** El botón es el elemento interactivo que el usuario presiona para realizar un "disparo".
    - **Label:**
        - **Nombre (Name):** `lblPuntaje`. Identificador de la etiqueta que muestra la puntuación.
        - **Propiedad `Text`:** Se actualiza para mostrar el puntaje actual del jugador (ej. "Puntaje: 1").
        - **Utilización:** Muestra el puntaje acumulado al usuario.
- **Variables de Soporte (No controles visuales):**
    - **`puntaje` (Integer):** Variable a nivel de clase que almacena la puntuación del jugador, inicializada a 0.

- **Explicación de la Lógica:**
    - Cuando se hace clic en `btnDisparar`, se ejecuta el evento `btnDisparar_Click`.
    - `MessageBox.Show("Acertaste?", "Disparo", MessageBoxButtons.YesNo)` despliega un cuadro de diálogo que simula una pregunta al usuario sobre si el disparo "acertó".
    - Si el usuario selecciona "Sí" (`DialogResult.Yes`), la variable `puntaje` se incrementa en 1 (`puntaje += 1`).
    - Finalmente, `lblPuntaje.Text` se actualiza para reflejar el `puntaje` actual.

Programa 106. Juego de laberinto con teclas de dirección

- **Explicación del Ejercicio:** Este programa es **idéntico al Programa 96**. Simula un juego de laberinto donde el usuario controla un "jugador" (representado por un botón) utilizando las teclas de dirección del teclado para moverlo y alcanzar una "meta" (otro botón).
- **Controles Necesarios:**
    - **Button (Jugador):**
        - **Nombre (Name):** `btnJugador`. Identificador del botón que representa al jugador.
        - **Propiedad `Text`:** "J". Identifica visualmente al jugador.
        - **Propiedad `Top`:** Representa la coordenada Y (posición vertical) del borde superior del botón. Se modifica para mover el jugador verticalmente.
        - **Propiedad `Left`:** Representa la coordenada X (posición horizontal) del borde izquierdo del botón. Se modifica para mover el jugador horizontalmente.
        - **Propiedad `Bounds`:** Define el tamaño y la ubicación del control. Es fundamental para la detección de colisiones con la meta (`btnMeta.Bounds`).
        - **Utilización:** Es el elemento visual interactivo que el usuario controla.
    - **Panel:**
        - **Nombre (Name):** `pnlLaberinto`. Aunque el código proporcionado no lo usa directamente para dibujar las paredes, su uso se sugiere como contenedor para el diseño del laberinto.
        - **Utilización:** Sirve como el área de juego principal.
    - **Button (Meta):**
        - **Nombre (Name):** `btnMeta`. Identificador del botón que representa el objetivo.
        - **Propiedad `Text`:** "M". Identifica visualmente la meta.
        - **Propiedad `Bounds`:** Se utiliza para determinar la posición y el tamaño de la meta para la detección de colisiones.
        - **Utilización:** Es el objetivo que el jugador debe alcanzar.
- **Configuración Adicional (del Formulario):**
    - **Propiedad `KeyPreview` (del Formulario):** Debe establecerse en `True`. Esto es crucial porque permite que el formulario capture los eventos de teclado (`KeyDown`)

antes que los controles individuales, asegurando que el movimiento del jugador funcione correctamente.

- **Explicación de la Lógica:**
  - El evento `Form1_KeyDown` (manejado por `Handles Me.KeyDown`) se dispara cada vez que el usuario presiona una tecla mientras el formulario tiene el foco.
  - Un bloque `Select Case e.KeyCode` evalúa qué tecla de dirección (`Keys.Up`, `Keys.Down`, `Keys.Left`, `Keys.Right`) fue presionada.
  - Dependiendo de la tecla, las propiedades `Top` o `Left` de `btnJugador` se ajustan en 5 píxeles (`-= 5` o `+= 5`), moviendo el botón del jugador en la dirección correspondiente.
  - Después de cada movimiento, `btnJugador.Bounds.IntersectsWith(btnMeta.Bounds)` verifica si los límites del `btnJugador` se superponen con los límites del `btnMeta`. Si se superponen, significa que el jugador ha alcanzado la meta, y se muestra un `MessageBox` con el mensaje "¡Ganaste!".

Programa 107. Agenda con búsqueda por nombre

- **Explicación del Ejercicio:** Este programa implementa una funcionalidad de **búsqueda para filtrar una lista de contactos** (en un `ListBox`) por nombre. Permite al usuario buscar contactos que contengan un texto específico, sin distinguir mayúsculas de minúsculas.
- **Controles Necesarios:**
  - **TextBox:**
    - **Nombre (Name):** `txtBuscar`. Campo de entrada donde el usuario escribe el texto a buscar.
    - **Propiedad `Text`:** Contiene la cadena de texto que el usuario desea usar como filtro. Se convierte a minúsculas (`.ToLower()`) para realizar una búsqueda insensible a mayúsculas.
    - **Utilización:** Es la interfaz para que el usuario ingrese su criterio de búsqueda.
  - **Button:**
    - **Nombre (Name):** `btnBuscar`. Identificador del botón.
    - **Texto (Text):** "Buscar". Texto visible en el botón.
    - **Evento `Click`:** `btnBuscar_Click`. Se activa al hacer clic. Contiene la lógica para filtrar la lista de contactos basándose en el texto de `txtBuscar`.
    - **Utilización:** Dispara la acción de buscar y filtrar los contactos.
  - **ListBox:**
    - **Nombre (Name):** `lstContactos`. Control para mostrar la lista de contactos.
    - **Colección `Items`:** Almacena y visualiza los nombres de los contactos. Se limpia y repuebla con los resultados del filtro.
    - **Método `Items.Clear()`:** Se llama antes de añadir los elementos filtrados para asegurar que solo se muestren los resultados de la búsqueda actual.
    - **Método `Items.AddRange()`:** En el evento `Form1_Load`, se utiliza para añadir los contactos iniciales del array `contactos` al `ListBox`.

        - **Método `Items.Add()`:** Se usa para añadir los contactos que cumplen con el criterio de búsqueda a la lista.
        - **Utilización:** Muestra la lista de contactos, filtrada según la búsqueda del usuario.
- **Variables de Soporte (No controles visuales):**
    - **`contactos` (List(Of String)):** Una lista (`New List(Of String)`) a nivel de clase que almacena la colección completa y original de nombres de contactos (ej., "Ana", "Luis", "Pedro"). Es la fuente de datos para la búsqueda.
- **Explicación de la Lógica:**
    - En el evento `Form1_Load`, la lista `contactos` se inicializa y sus elementos se añaden a `lstContactos` para que se muestren al inicio.
    - Cuando se hace clic en `btnBuscar`, el texto de `txtBuscar` se obtiene y se convierte a minúsculas, almacenándose en la variable `filtro`.
    - `lstContactos.Items.Clear()` vacía la lista visible de contactos.
    - Un bucle `For Each` itera sobre la lista original `contactos`.
    - Dentro del bucle, `c.ToLower().Contains(filtro)` verifica si el nombre del contacto actual (convertido a minúsculas) contiene el texto del `filtro`.
    - Si hay una coincidencia, el contacto `c` se añade de nuevo a `lstContactos`.

Programa 108. Agenda con edición de contactos

- **Explicación del Ejercicio:** Este programa permite al usuario **editar el texto de un contacto ya existente** que esté seleccionado en un `ListBox`, actualizando su contenido con el texto ingresado en un `TextBox`.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtContacto`. Campo de entrada donde el usuario escribe el nuevo texto para el contacto.
        - **Propiedad `Text`:** Contiene el texto que se utilizará para actualizar el elemento seleccionado en el `ListBox`.
        - **Utilización:** Es la interfaz para que el usuario escriba la nueva información del contacto.
    - **Button:**
        - **Nombre (Name):** `btnEditar`. Identificador del botón.
        - **Texto (Text):** "Editar". Texto visible en el botón.
        - **Evento `Click`:** `btnEditar_Click`. Se activa cuando el usuario pulsa el botón. Contiene la lógica para modificar el elemento seleccionado en el `ListBox`.
        - **Utilización:** Dispara la acción de edición del contacto.
    - **ListBox:**
        - **Nombre (Name):** `lstContactos`. Control que muestra la lista de contactos.
        - **Colección `Items`:** Almacena y visualiza los elementos de la lista.
        - **Propiedad `SelectedIndex`:** Devuelve el índice (posición) del elemento que está actualmente seleccionado por el usuario. Su valor es `-1` si no hay

ningún elemento seleccionado. Es crucial para identificar qué elemento se va a modificar.
        - **Método `Items(lstContactos.SelectedIndex)`:** Permite acceder a un elemento específico de la colección `Items` por su índice.
        - **Utilización:** Muestra la lista de contactos y permite al usuario seleccionar uno para editar.
- **Explicación de la Lógica:**
    - Cuando se hace clic en `btnEditar`, el evento `btnEditar_Click` se ejecuta.
    - Se verifica si hay un elemento seleccionado en `lstContactos` mediante la condición `If lstContactos.SelectedIndex >= 0 Then`. Esto evita errores si el usuario intenta editar sin haber seleccionado nada.
    - Si hay un elemento seleccionado, `lstContactos.Items(lstContactos.SelectedIndex) = txtContacto.Text` reemplaza el texto del elemento en la posición seleccionada con el nuevo texto del `txtContacto.Text`. Esto actualiza visualmente el contacto en la lista.

Programa 109. Agenda con borrado de contactos

- **Explicación del Ejercicio:** Este programa permite al usuario **eliminar un contacto seleccionado** de un `ListBox`. Es una funcionalidad básica para la gestión de listas.
- **Controles Necesarios:**
    - **Button:**
        - **Nombre (Name):** `btnBorrar`. Identificador del botón.
        - **Texto (Text):** "Borrar". Texto visible en el botón.
        - **Evento `Click`:** `btnBorrar_Click`. Se activa cuando el usuario pulsa el botón. Contiene la lógica para eliminar el elemento seleccionado del `ListBox`.
        - **Utilización:** Dispara la acción de eliminar un contacto.
    - **ListBox:**
        - **Nombre (Name):** `lstContactos`. Control que muestra la lista de contactos.
        - **Colección `Items`:** Almacena y visualiza los elementos de la lista.
        - **Propiedad `SelectedIndex`:** Devuelve el índice (posición) del elemento que está actualmente seleccionado por el usuario. Su valor es `-1` si no hay ningún elemento seleccionado. Es crucial para identificar qué elemento se va a eliminar.
        - **Método `Items.RemoveAt(lstContactos.SelectedIndex)`:** Elimina el elemento de la colección `Items` en la posición especificada por `SelectedIndex`.
        - **Utilización:** Muestra la lista de contactos y permite al usuario seleccionar uno para borrar.
- **Explicación de la Lógica:**
    - Cuando se hace clic en `btnBorrar`, el evento `btnBorrar_Click` se ejecuta.

- Se verifica si hay un elemento seleccionado en `lstContactos` mediante la condición `If lstContactos.SelectedIndex >= 0 Then`. Esto evita errores si el usuario intenta borrar sin haber seleccionado nada.
- Si hay un elemento seleccionado, `lstContactos.Items.RemoveAt(lstContactos.SelectedIndex)` elimina el elemento en la posición indicada por `SelectedIndex`.

Programa 110. Agenda con exportación a archivo de texto

- **Explicación del Ejercicio:** Este programa permite al usuario **exportar la lista de contactos** que se encuentran en un `ListBox` a un archivo de texto plano (`agenda.txt`).
- **Controles Necesarios:**
  - **Button:**
    - **Nombre (Name):** `btnExportar`. Identificador del botón.
    - **Texto (Text):** "Exportar". Texto visible en el botón.
    - **Evento `Click`:** `btnExportar_Click`. Se activa cuando el usuario pulsa el botón. Contiene la lógica para escribir cada elemento del `ListBox` en un archivo de texto.
    - **Utilización:** Dispara la acción de exportar los contactos a un archivo.
  - **ListBox:**
    - **Nombre (Name):** `lstContactos`. Control que muestra la lista de contactos.
    - **Colección `Items`:** Contiene los elementos (cadenas de texto) que se exportarán al archivo.
    - **Utilización:** Almacena los contactos que serán exportados.
- **Importación de Namespace:**
  - **`Imports System.IO`:** Es necesario para utilizar las clases y métodos relacionados con la manipulación de archivos, como `StreamWriter`.
- **Clases y Métodos de Archivos:**
  - **`StreamWriter` (`sw`):** Una clase utilizada para escribir texto en un archivo.
  - **`Using sw As New StreamWriter("agenda.txt")`:** Esta declaración crea una nueva instancia de `StreamWriter` y la abre para escribir en el archivo "agenda.txt". El bloque `Using` garantiza que el `StreamWriter` se cierre correctamente y los recursos se liberen automáticamente al finalizar, incluso si ocurre un error.
  - **`sw.WriteLine(c)`:** Este método escribe una línea de texto (`c`, que es un elemento del `ListBox`) en el archivo, seguido de un salto de línea.
- **Explicación de la Lógica:**
  - Cuando se hace clic en `btnExportar`, se ejecuta el evento `btnExportar_Click`.
  - Se crea un `StreamWriter` para el archivo "agenda.txt".
  - Un bucle `For Each` itera sobre cada elemento (`c` de tipo `String`) en la colección `lstContactos.Items`.
  - En cada iteración, `sw.WriteLine(c)` escribe el elemento actual en el archivo, asegurando que cada contacto ocupe una nueva línea.

- Después de que el bucle termina y el `StreamWriter` se cierra (automáticamente por el bloque `Using`), un `MessageBox` muestra un mensaje de confirmación "Exportado correctamente".

Programa 111. Agenda con importación desde archivo de texto

- **Explicación del Ejercicio:** Este programa permite al usuario **importar contactos desde un archivo de texto** existente y cargarlos en un `ListBox`.
- **Controles Necesarios:**
  - **Button:**
    - **Nombre (Name):** `btnImportar`. Identificador del botón.
    - **Texto (Text):** "Importar". Texto visible en el botón.
    - **Evento `Click`:** `btnImportar_Click`. Se activa cuando el usuario pulsa el botón. Contiene la lógica para abrir el `OpenFileDialog` y cargar los contactos.
    - **Utilización:** Dispara la acción de importar contactos desde un archivo.
  - **ListBox:**
    - **Nombre (Name):** `lstContactos`. Control que muestra la lista de contactos.
    - **Colección `Items`:** Almacena y visualiza los elementos de la lista. Se limpia antes de la importación y luego se repuebla con los elementos del archivo.
    - **Método `Items.Clear()`:** Se llama para eliminar cualquier contacto existente en la lista antes de cargar los nuevos, asegurando que solo se muestren los datos importados.
    - **Método `Items.AddRange(lineas)`:** Añade todos los elementos de un array de cadenas (`lineas`) a la colección `Items` del `ListBox` de una sola vez.
    - **Utilización:** Almacena y visualiza los contactos importados.
  - **OpenFileDialog:**
    - **Nombre (Name):** `OpenFileDialog1`. Componente no visible que muestra el cuadro de diálogo estándar de Windows para seleccionar un archivo.
    - **Método `ShowDialog()`:** Abre el cuadro de diálogo de selección de archivo. Retorna `DialogResult.OK` si el usuario selecciona un archivo y hace clic en "Abrir".
    - **Propiedad `FileName`:** Después de que el usuario selecciona un archivo y lo confirma, esta propiedad contiene la ruta completa del archivo elegido.
    - **Utilización:** Facilita la interacción del usuario para seleccionar el archivo de contactos a importar.
- **Importación de Namespace:**
  - **`Imports System.IO`:** Es necesario para utilizar los métodos de lectura de archivos.
- **Clases y Métodos de Archivos:**
  - **`File.ReadAllLines(OpenFileDialog1.FileName)`:** Este método estático de la clase `File` lee todas las líneas de texto de un archivo especificado (por

su ruta `OpenFileDialog1.FileName`) y las devuelve como un array de cadenas. Cada elemento del array es una línea del archivo.

- **Explicación de la Lógica:**
    - Al hacer clic en `btnImportar`, `OpenFileDialog1.ShowDialog()` se abre para que el usuario elija un archivo.
    - Si el usuario selecciona un archivo y presiona "Abrir" (`DialogResult.OK`), la condición `If OpenFileDialog1.ShowDialog() = DialogResult.OK Then` se cumple.
    - Dentro de la condición, `lstContactos.Items.Clear()` limpia la lista actual de contactos.
    - `Dim lineas() As String = File.ReadAllLines(OpenFileDialog1.FileName)` lee todas las líneas del archivo seleccionado y las almacena en el array `lineas`.
    - `lstContactos.Items.AddRange(lineas)` añade todos los elementos del array `lineas` al `ListBox`, mostrando los contactos importados.

Programa 112. Agenda con exportación a Excel

- **Explicación del Ejercicio:** Este programa permite **exportar la lista de contactos** desde un `ListBox` a una hoja de cálculo de **Microsoft Excel**. Esto implica interactuar con la aplicación Excel a través de su modelo de objetos.
- **Controles Necesarios:**
    - **Button:**
        - **Nombre (Name):** `btnExportarExcel`. Identificador del botón.
        - **Texto (Text):** "Exportar a Excel". Texto visible en el botón.
        - **Evento `Click`:** `btnExportarExcel_Click`. Se activa cuando el usuario pulsa el botón. Contiene la lógica para abrir Excel, crear un nuevo libro de trabajo y escribir los datos del `ListBox` en sus celdas.
        - **Utilización:** Dispara la acción de exportar los contactos a Excel.
    - **ListBox:**
        - **Nombre (Name):** `lstContactos`. Control que contiene los elementos a exportar.
        - **Colección `Items`:** Contiene las cadenas de texto que se escribirán en las celdas de Excel.
        - **Utilización:** Proporciona la fuente de datos para la exportación.
- **Requisito Adicional (No es un control visual):**
    - **Referencia a la biblioteca de objetos de Microsoft Excel:** Este programa **requiere agregar una referencia** a la biblioteca de objetos de Microsoft Excel en el proyecto de Visual Studio. Esto se hace típicamente en `Project > Add Reference > COM > Microsoft Excel Object Library`.
- **Importación de Namespace:**
    - **`Imports Microsoft.Office.Interop.Excel`:** Es esencial para acceder a las clases y objetos del modelo de objetos de Excel, como `Application`, `Workbook` y `Worksheet`.
- **Clases y Métodos de Excel Interop:**

        - **`Application` (`app`):** Representa la aplicación de Excel en sí.
        - **`app.Workbooks.Add()`:** Crea un nuevo libro de trabajo vacío en Excel (`wb`).
        - **`wb.Sheets(1)`:** Accede a la primera hoja de cálculo del libro de trabajo.
        - **`CType(wb.Sheets(1), Worksheet)`:** Realiza un *cast* explícito para tratar el objeto `Sheet` como un `Worksheet`, lo que permite acceder a sus métodos y propiedades específicos.
        - **`ws.Cells(i + 1, 1)`:** Representa una celda específica en la hoja de trabajo. `i + 1` es el número de fila (ajustado porque los índices del bucle comienzan en 0), y `1` es la primera columna.
- **Explicación de la Lógica:**
    - Cuando se hace clic en `btnExportarExcel`, se ejecuta el evento `btnExportarExcel_Click`.
    - `Dim app As New Application` crea una nueva instancia oculta de la aplicación Microsoft Excel.
    - `Dim wb = app.Workbooks.Add()` añade un nuevo libro de trabajo a esta instancia de Excel.
    - `Dim ws = CType(wb.Sheets(1), Worksheet)` obtiene una referencia a la primera hoja de trabajo del libro recién creado.
    - Un bucle `For i As Integer = 0 To lstContactos.Items.Count - 1` itera sobre cada elemento en el `lstContactos`.
    - Dentro del bucle, `ws.Cells(i + 1, 1) = lstContactos.Items(i)` escribe el texto de cada elemento del `ListBox` en la celda correspondiente de la primera columna de Excel.
    - Finalmente, `app.Visible = True` hace visible la aplicación Excel para el usuario, mostrando los datos exportados.

Programa 113. Agenda con importación desde Excel

- **Explicación del Ejercicio:** Este programa permite al usuario **importar contactos desde una hoja de cálculo de Microsoft Excel** a un `ListBox`. Requiere que el usuario seleccione un archivo Excel mediante un `OpenFileDialog`.
- **Controles Necesarios:**
    - **Button:**
        - **Nombre (Name):** `btnImportarExcel`. Identificador del botón.
        - **Texto (Text):** "Importar desde Excel". Texto visible en el botón.
        - **Evento `Click`:** `btnImportarExcel_Click`. Se activa cuando el usuario pulsa el botón. Contiene la lógica para abrir Excel, cargar el libro de trabajo seleccionado y leer los datos en el `ListBox`.
        - **Utilización:** Dispara la acción de importar los contactos desde un archivo Excel.
    - **ListBox:**
        - **Nombre (Name):** `lstContactos`. Control que muestra la lista de contactos.
        - **Colección `Items`:** Se limpia antes de la importación y luego se repuebla con los datos leídos de Excel.

        - **Método `Items.Clear()`:** Vacía el `ListBox` antes de cargar nuevos datos para evitar duplicados y mostrar solo los datos importados.
        - **Método `Items.Add()`:** Añade cada valor leído de una celda de Excel como un nuevo elemento a la lista.
        - **Utilización:** Almacena y visualiza los contactos importados.
    - **OpenFileDialog:**
        - **Nombre (Name):** `OpenFileDialog1`. Componente no visible que muestra el cuadro de diálogo estándar de Windows para seleccionar un archivo.
        - **Método `ShowDialog()`:** Abre el cuadro de diálogo de selección de archivo. Retorna `DialogResult.OK` si el usuario selecciona un archivo y hace clic en "Abrir".
        - **Propiedad `FileName`:** Contiene la ruta completa del archivo Excel seleccionado por el usuario.
        - **Utilización:** Facilita la interacción del usuario para seleccionar el archivo Excel a importar.
- **Requisito Adicional (No es un control visual):**
    - **Referencia a la biblioteca de objetos de Microsoft Excel:** Este programa **requiere agregar una referencia** a la biblioteca de objetos de Microsoft Excel en el proyecto.
- **Importación de Namespace:**
    - **`Imports Microsoft.Office.Interop.Excel`:** Es esencial para interactuar con la aplicación Excel.
- **Clases y Métodos de Excel Interop:**
    - **`Application` (`app`):** Representa la aplicación de Excel.
    - **`app.Workbooks.Open(OpenFileDialog1.FileName)`:** Abre un libro de trabajo existente especificado por la ruta del archivo.
    - **`wb.Sheets(1)`:** Accede a la primera hoja de cálculo.
    - **`ws.Cells(fila, 1).Value`:** Lee el valor de una celda específica en la hoja de trabajo. `fila` es el número de fila actual, y `1` es la primera columna.
    - **`wb.Close(False)`:** Cierra el libro de trabajo. `False` indica que no se guarden los cambios.
    - **`app.Quit()`:** Cierra la aplicación de Excel que se abrió programáticamente.
- **Explicación de la Lógica:**
    - Cuando se hace clic en `btnImportarExcel`, `OpenFileDialog1.ShowDialog()` se abre.
    - Si el usuario selecciona un archivo y presiona "Abrir" (`DialogResult.OK`), se ejecuta el bloque de código.
    - `Dim app As New Application` crea una nueva instancia oculta de Excel.
    - `Dim wb = app.Workbooks.Open(OpenFileDialog1.FileName)` abre el libro de trabajo de Excel que el usuario seleccionó.
    - `Dim ws = CType(wb.Sheets(1), Worksheet)` obtiene una referencia a la primera hoja de trabajo.
    - `lstContactos.Items.Clear()` limpia el `ListBox`.

- Un bucle `While ws.Cells(fila, 1).Value IsNot Nothing` lee los datos fila por fila de la primera columna. El bucle continúa mientras las celdas no estén vacías.
- `lstContactos.Items.Add(ws.Cells(fila, 1).Value)` añade el valor de cada celda a `lstContactos`.
- `fila += 1` incrementa el contador de fila.
- Después de leer todos los datos, `wb.Close(False)` cierra el libro de trabajo sin guardar cambios, y `app.Quit()` cierra la instancia de Excel.

Programa 114. Agenda con base de datos SQLite

- **Explicación del Ejercicio:** Este programa utiliza una **base de datos SQLite** para almacenar y gestionar contactos (específicamente, los nombres de los contactos). Permite agregar nuevos contactos y cargar los existentes en un `ListBox`.
- **Controles Necesarios:**
  - **Button:**
    - **Nombre (Name):** `btnCargar`. Identificador del botón.
    - **Texto (Text):** "Cargar". Texto visible en el botón.
    - **Utilización:** Este botón se utilizaría para recargar explícitamente la lista de contactos desde la base de datos, aunque la carga inicial ya se realiza en `Form1_Load`.
  - **Button:**
    - **Nombre (Name):** `btnAgregar`. Identificador del botón.
    - **Texto (Text):** "Agregar". Texto visible en el botón.
    - **Evento `Click`:** `btnAgregar_Click`. Se activa al hacer clic. Contiene la lógica para insertar un nuevo contacto en la base de datos y luego recargar la lista en el `ListBox`.
    - **Utilización:** Permite al usuario añadir un nuevo contacto a la agenda.
  - **TextBox:**
    - **Nombre (Name):** `txtContacto`. Campo de entrada para el nombre del contacto.
    - **Propiedad `Text`:** Contiene el nombre del contacto que se va a agregar a la base de datos. Se utiliza como un parámetro en la consulta SQL.
    - **Utilización:** Permite al usuario introducir el nombre del nuevo contacto.
  - **ListBox:**
    - **Nombre (Name):** `lstContactos`. Control para mostrar la lista de contactos.
    - **Colección `Items`:** Se limpia antes de cargar los contactos de la base de datos y luego se repuebla con los resultados de la consulta.
    - **Método `Items.Clear()`:** Limpia el `ListBox` de cualquier elemento anterior.
    - **Método `Items.Add(dr("nombre"))`:** Añade el valor de la columna "nombre" de la fila actual del `DataReader` al `ListBox`.
    - **Utilización:** Muestra la lista de contactos almacenados en la base de datos.
- **Requisito Adicional (No es un control visual):**

    - **Librería `System.Data.SQLite`:** Este programa **requiere agregar una referencia** a la librería `System.Data.SQLite`, que generalmente se instala a través del administrador de paquetes NuGet en Visual Studio.
- **Importación de Namespace:**
    - **`Imports System.Data.SQLite`:** Es fundamental para utilizar las clases de conectividad a bases de datos SQLite.
- **Clases y Métodos de SQLite:**
    - **`SQLiteConnection (conn)`:** Objeto a nivel de clase que representa la conexión a la base de datos SQLite. Se inicializa con una cadena de conexión que especifica la ruta del archivo de la base de datos (`Data Source=agenda.db;`) y la versión (`Version=3;`).
    - **`conn.Open()`:** Establece la conexión con la base de datos.
    - **`SQLiteCommand (cmd)`:** Objeto que representa un comando SQL a ejecutar.
    - **`cmd.ExecuteNonQuery()`:** Ejecuta un comando SQL que no devuelve filas (como `CREATE TABLE` o `INSERT`).
    - **`cmd.ExecuteReader()`:** Ejecuta un comando SQL que devuelve filas (como `SELECT`) y retorna un `SQLiteDataReader`.
    - **`SQLiteDataReader (dr)`:** Se utiliza para leer los resultados fila por fila de una consulta `SELECT`.
    - **`dr.Read()`:** Avanza el `DataReader` a la siguiente fila. Retorna `True` si hay más filas, `False` si ha llegado al final.
    - **`cmd.Parameters.AddWithValue("@nombre", txtContacto.Text)`:** Añade un parámetro a la consulta SQL, lo cual es una buena práctica para prevenir la inyección SQL. `@nombre` es el nombre del parámetro en la consulta, y `txtContacto.Text` es su valor.
- **Explicación de la Lógica:**
    - La conexión a la base de datos (`conn`) se declara a nivel de clase.
    - En el evento `Form1_Load`:
        - `conn.Open()` abre la conexión a `agenda.db`.
        - Se crea un comando SQL (`CREATE TABLE IF NOT EXISTS...`) para asegurarse de que la tabla `contactos` exista. Si no existe, se crea con una columna `nombre` de tipo `TEXT`.
        - `CargarContactos()` se llama para poblar `lstContactos` con los datos existentes al inicio de la aplicación.
    - El método `CargarContactos()`:
        - Limpia `lstContactos`.
        - Crea un `SQLiteCommand` para seleccionar todos los `nombre` de la tabla `contactos`.
        - `cmd.ExecuteReader()` ejecuta la consulta y devuelve un `DataReader`.
        - Un bucle `While dr.Read()` itera sobre cada fila de resultados, añadiendo el `nombre` de cada contacto a `lstContactos`.
        - `(dr.Close()` se sugiere para robustez).

- En el evento `btnAgregar_Click`:
    - Se crea un `SQLiteCommand` con una consulta `INSERT` parametrizada para añadir un nuevo contacto a la tabla `contactos`.
    - `cmd.Parameters.AddWithValue()` asigna el texto de `txtContacto` al parámetro `@nombre`.
    - `cmd.ExecuteNonQuery()` ejecuta la inserción del contacto.
    - `CargarContactos()` se llama nuevamente para actualizar el `lstContactos` con el nuevo elemento.

Programa 115. Agenda con base de datos SQL Server

- **Explicación del Ejercicio:** Este programa integra una **base de datos SQL Server** para almacenar y gestionar contactos. Permite agregar nuevos contactos y cargar los existentes en un `ListBox`, utilizando el proveedor de datos `SqlClient`.
- **Controles Necesarios:**
    - **Button:**
        - **Nombre (Name):** `btnCargar`. Identificador del botón.
        - **Texto (Text):** "Cargar". Texto visible en el botón.
        - **Utilización:** Este botón se utilizaría para recargar explícitamente la lista de contactos desde la base de datos.
    - **Button:**
        - **Nombre (Name):** `btnAgregar`. Identificador del botón.
        - **Texto (Text):** "Agregar". Texto visible en el botón.
        - **Evento `Click`:** `btnAgregar_Click`. Contiene la lógica para insertar un nuevo contacto en SQL Server y actualizar el `ListBox`.
        - **Utilización:** Permite al usuario añadir un nuevo contacto.
    - **TextBox:**
        - **Nombre (Name):** `txtContacto`. Campo de entrada para el nombre del contacto.
        - **Propiedad `Text`:** Contiene el nombre del contacto a insertar. Se usa como parámetro en la consulta SQL.
        - **Utilización:** Permite al usuario introducir el nombre del nuevo contacto.
    - **ListBox:**
        - **Nombre (Name):** `lstContactos`. Control para mostrar la lista de contactos.
        - **Colección `Items`:** Se limpia antes de la carga y se repuebla con los resultados de la base de datos.
        - **Método `Items.Clear()`:** Limpia el `ListBox`.
        - **Método `Items.Add(dr("nombre"))`:** Añade el valor de la columna "nombre" del `DataReader` al `ListBox`.
        - **Utilización:** Muestra los contactos almacenados en la base de datos.
- **Requisito Adicional (No es un control visual):**
    - **Servidor SQL Server:** Se requiere tener un servidor SQL Server instalado y una base de datos llamada "Agenda" con una tabla "contactos" que tenga una columna "nombre".

- **Importación de Namespace:**
  - **`Imports System.Data.SqlClient`:** Es fundamental para interactuar con bases de datos SQL Server.
- **Clases y Métodos de SQLClient:**
  - **`SqlConnection` (`conn`):** Objeto a nivel de clase que representa la conexión a la base de datos SQL Server. La cadena de conexión especifica el servidor (`Server=localhost;`), la base de datos (`Database=Agenda;`) y el tipo de autenticación (`Trusted_Connection=True;` para autenticación de Windows).
  - **`conn.Open()`:** Abre la conexión.
  - **`SqlCommand` (`cmd`):** Objeto que representa un comando SQL.
  - **`cmd.ExecuteNonQuery()`:** Ejecuta consultas que no devuelven datos (como `INSERT`).
  - **`cmd.ExecuteReader()`:** Ejecuta consultas que devuelven un conjunto de resultados (`SELECT`).
  - **`SqlDataReader` (`dr`):** Lee los resultados de `SELECT`.
  - **`dr.Read()`:** Avanza a la siguiente fila.
  - **`dr.Close()`:** Cierra el `DataReader`. Es importante liberar este recurso.
  - **`cmd.Parameters.AddWithValue("@nombre", txtContacto.Text)`:** Añade un parámetro a la consulta SQL.
- **Explicación de la Lógica:**
  - La conexión a la base de datos (`conn`) se declara a nivel de clase.
  - En `Form1_Load`:
    - `conn.Open()` abre la conexión.
    - `CargarContactos()` se llama para poblar `lstContactos`.
  - El método `CargarContactos()`:
    - Limpia `lstContactos`.
    - Crea un `SqlCommand` para `SELECT` los nombres.
    - `cmd.ExecuteReader()` ejecuta la consulta, obteniendo un `DataReader`.
    - Un bucle `While dr.Read()` itera sobre los resultados y añade cada nombre a `lstContactos`.
    - `dr.Close()` cierra explícitamente el `DataReader`.
  - En `btnAgregar_Click`:
    - Se crea un `SqlCommand` con una consulta `INSERT` parametrizada.
    - Se asigna el texto de `txtContacto` al parámetro `@nombre`.
    - `cmd.ExecuteNonQuery()` ejecuta la inserción.
    - `CargarContactos()` se llama para actualizar la lista en el `ListBox`.

Programa 116. Agenda con base de datos MySQL

- **Explicación del Ejercicio:** Este programa utiliza una **base de datos MySQL** para almacenar y gestionar contactos. Permite agregar nuevos contactos y cargar los existentes en un `ListBox`, empleando el conector `MySqlClient`.
- **Controles Necesarios:**

    - **Button:**
        - **Nombre (Name):** `btnCargar`. Identificador del botón.
        - **Texto (Text):** "Cargar". Texto visible en el botón.
        - **Utilización:** Se usaría para cargar explícitamente los contactos.
    - **Button:**
        - **Nombre (Name):** `btnAgregar`. Identificador del botón.
        - **Texto (Text):** "Agregar". Texto visible en el botón.
        - **Evento `Click`:** `btnAgregar_Click`. Contiene la lógica para insertar un nuevo contacto en MySQL y actualizar el `ListBox`.
        - **Utilización:** Permite añadir nuevos contactos a la agenda.
    - **TextBox:**
        - **Nombre (Name):** `txtContacto`. Campo de entrada para el nombre del contacto.
        - **Propiedad `Text`:** Contiene el nombre del contacto a insertar, usado como parámetro.
        - **Utilización:** Permite introducir el nombre del nuevo contacto.
    - **ListBox:**
        - **Nombre (Name):** `lstContactos`. Control para mostrar la lista de contactos.
        - **Colección `Items`:** Se limpia antes de la carga y se repuebla con los resultados de la base de datos.
        - **Método `Items.Clear()`:** Limpia el `ListBox`.
        - **Método `Items.Add(dr("nombre"))`:** Añade el valor de la columna "nombre" del `DataReader` al `ListBox`.
        - **Utilización:** Muestra los contactos almacenados en la base de datos MySQL.
- **Requisito Adicional (No es un control visual):**
    - **Conector `MySql.Data`:** Este programa **requiere agregar una referencia** a la librería `MySql.Data`, que se instala vía NuGet. También se necesita un servidor MySQL con una base de datos "agenda" y una tabla "contactos" (con una columna `nombre`).
- **Importación de Namespace:**
    - **`Imports MySql.Data.MySqlClient`:** Es fundamental para interactuar con bases de datos MySQL.
- **Clases y Métodos de MySqlClient:**
    - **`MySqlConnection` (`conn`):** Objeto a nivel de clase que representa la conexión a la base de datos MySQL. La cadena de conexión incluye el servidor (`server=localhost;`), usuario (`user=root;`), base de datos (`database=agenda;`) y contraseña (`password=;`).
    - **`conn.Open()`:** Abre la conexión.
    - **`MySqlCommand` (`cmd`):** Objeto para comandos SQL.
    - **`cmd.ExecuteNonQuery()`:** Ejecuta consultas que no devuelven datos.
    - **`cmd.ExecuteReader()`:** Ejecuta consultas que devuelven un conjunto de resultados.
    - **`MySqlDataReader` (`dr`):** Lee los resultados de `SELECT`.

        - **`dr.Read()`:** Avanza a la siguiente fila.
        - **`dr.Close()`:** Cierra el `DataReader`.
        - **`cmd.Parameters.AddWithValue("@nombre", txtContacto.Text)`:** Añade un parámetro a la consulta SQL.
- **Explicación de la Lógica:**
    - La conexión a MySQL (`conn`) se declara a nivel de clase.
    - En `Form1_Load`:
        - `conn.Open()` abre la conexión.
        - `CargarContactos()` se llama para poblar `lstContactos`.
    - El método `CargarContactos()`:
        - Limpia `lstContactos`.
        - Crea un `MySqlCommand` para `SELECT` los nombres.
        - `cmd.ExecuteReader()` ejecuta la consulta, obteniendo un `DataReader`.
        - Un bucle `While dr.Read()` itera sobre los resultados y añade cada nombre a `lstContactos`.
        - `dr.Close()` cierra el `DataReader`.
    - En `btnAgregar_Click`:
        - Se crea un `MySqlCommand` con una consulta `INSERT` parametrizada.
        - Se asigna el texto de `txtContacto` al parámetro `@nombre`.
        - `cmd.ExecuteNonQuery()` ejecuta la inserción.
        - `CargarContactos()` se llama para actualizar la lista.

Programa 117. Agenda con base de datos PostgreSQL

- **Explicación del Ejercicio:** Este programa utiliza una **base de datos PostgreSQL** para almacenar y gestionar contactos. Permite agregar nuevos contactos y cargar los existentes en un `ListBox`, utilizando el conector `Npgsql`.
- **Controles Necesarios:**
    - **Button:**
        - **Nombre (Name):** `btnCargar`. Identificador del botón.
        - **Texto (Text):** "Cargar". Texto visible en el botón.
        - **Utilización:** Se usaría para cargar explícitamente los contactos.
    - **Button:**
        - **Nombre (Name):** `btnAgregar`. Identificador del botón.
        - **Texto (Text):** "Agregar". Texto visible en el botón.
        - **Evento `Click`:** `btnAgregar_Click`. Contiene la lógica para insertar un nuevo contacto en PostgreSQL y actualizar el `ListBox`.
        - **Utilización:** Permite añadir nuevos contactos a la agenda.
    - **TextBox:**
        - **Nombre (Name):** `txtContacto`. Campo de entrada para el nombre del contacto.
        - **Propiedad `Text`:** Contiene el nombre del contacto a insertar, usado como parámetro.
        - **Utilización:** Permite introducir el nombre del nuevo contacto.

    - **ListBox:**
        - **Nombre (Name):** `lstContactos`. Control para mostrar la lista de contactos.
        - **Colección `Items`:** Se limpia antes de la carga y se repuebla con los resultados de la base de datos.
        - **Método `Items.Clear()`:** Limpia el `ListBox`.
        - **Método `Items.Add(dr("nombre"))`:** Añade el valor de la columna "nombre" del `DataReader` al `ListBox`.
        - **Utilización:** Muestra los contactos almacenados en la base de datos PostgreSQL.
- **Requisito Adicional (No es un control visual):**
    - **Conector `Npgsql`:** Este programa **requiere agregar una referencia** a la librería `Npgsql`, que se instala vía NuGet. También se necesita un servidor PostgreSQL con una base de datos "agenda" y una tabla "contactos" (con una columna `nombre`).
- **Importación de Namespace:**
    - **`Imports Npgsql`:** Es fundamental para interactuar con bases de datos PostgreSQL.
- **Clases y Métodos de Npgsql:**
    - **`NpgsqlConnection` (`conn`):** Objeto a nivel de clase que representa la conexión a la base de datos PostgreSQL. La cadena de conexión incluye el host, usuario, base de datos y contraseña.
    - **`conn.Open()`:** Abre la conexión.
    - **`NpgsqlCommand` (`cmd`):** Objeto para comandos SQL.
    - **`cmd.ExecuteNonQuery()`:** Ejecuta consultas que no devuelven datos.
    - **`cmd.ExecuteReader()`:** Ejecuta consultas que devuelven un conjunto de resultados.
    - **`NpgsqlDataReader` (`dr`):** Lee los resultados de `SELECT`.
    - **`dr.Read()`:** Avanza a la siguiente fila.
    - **`dr.Close()`:** Cierra el `DataReader`.
    - **`cmd.Parameters.AddWithValue("@nombre", txtContacto.Text)`:** Añade un parámetro a la consulta SQL.
- **Explicación de la Lógica:**
    - La conexión a PostgreSQL (`conn`) se declara a nivel de clase.
    - En `Form1_Load`:
        - `conn.Open()` abre la conexión.
        - `CargarContactos()` se llama para poblar `lstContactos`.
    - El método `CargarContactos()`:
        - Limpia `lstContactos`.
        - Crea un `NpgsqlCommand` para `SELECT` los nombres.
        - `cmd.ExecuteReader()` ejecuta la consulta, obteniendo un `DataReader`.
        - Un bucle `While dr.Read()` itera sobre los resultados y añade cada nombre a `lstContactos`.
        - `dr.Close()` cierra el `DataReader`.

- En `btnAgregar_Click`:
    - Se crea un `NpgsqlCommand` con una consulta `INSERT` parametrizada.
    - Se asigna el texto de `txtContacto` al parámetro `@nombre`.
    - `cmd.ExecuteNonQuery()` ejecuta la inserción.
    - `CargarContactos()` se llama para actualizar la lista.

Programa 118. Aplicación de notas con recordatorio

- **Explicación del Ejercicio:** Este programa permite al usuario **guardar notas y establecer una fecha y hora específicas como recordatorio**. Cuando la fecha y hora de un recordatorio se cumplen, la aplicación muestra un `MessageBox` con el texto de la nota.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtNota`. Campo de entrada para el texto de la nota.
        - **Propiedad `Multiline`:** `True`. Permite la entrada de varias líneas de texto para notas más extensas.
        - **Propiedad `Text`:** Contiene el texto que el usuario desea guardar como nota y que aparecerá en el recordatorio.
        - **Utilización:** Es el área donde el usuario escribe el contenido de la nota.
    - **Button:**
        - **Nombre (Name):** `btnGuardar`. Identificador del botón.
        - **Texto (Text):** "Guardar Nota". Texto visible en el botón.
        - **Evento `Click`:** `btnGuardar_Click`. Se activa al hacer clic. Contiene la lógica para añadir la nota y su fecha/hora de recordatorio a una lista.
        - **Utilización:** Dispara la acción de guardar la nota y configurar el recordatorio.
    - **DateTimePicker:**
        - **Nombre (Name):** `dtpRecordatorio`. Control que permite al usuario seleccionar una fecha y hora.
        - **Propiedad `Value`:** Contiene la fecha y hora seleccionadas por el usuario. Se utiliza para establecer la hora del recordatorio.
        - **Utilización:** Proporciona una interfaz amigable para seleccionar la fecha y hora del recordatorio.
    - **Timer:**
        - **Nombre (Name):** `Timer1`. Componente no visible que genera eventos en intervalos regulares.
        - **Propiedad `Interval`:** 1000. Se configura para disparar su evento `Tick` cada 1000 milisegundos (1 segundo).
        - **Propiedad `Enabled`:** `True` (se asume que se habilita al inicio para que el temporizador esté activo, aunque no se especifica en la fuente).
        - **Evento `Tick`:** `Timer1_Tick`. En cada disparo, itera sobre las notas guardadas para comprobar si alguna fecha/hora de recordatorio se ha cumplido.
        - **Utilización:** Es el motor que verifica continuamente si alguna alarma debe dispararse.
- **Variables y Clases de Soporte (No controles visuales):**

        - **`notas` (List(Of Tuple(Of String, DateTime))):** Una lista a nivel de clase que almacena las notas. Cada elemento de la lista es una `Tuple` que contiene dos valores: el texto de la nota (`String`) y la fecha/hora del recordatorio (`DateTime`). `Tuple` es útil para agrupar datos relacionados.
- **Explicación de la Lógica:**
    - La lista `notas` se declara a nivel de clase para almacenar las tuplas (nota, recordatorio).
    - En el evento `btnGuardar_Click`:
        - El texto de `txtNota` y la fecha/hora seleccionada en `dtpRecordatorio.Value` se combinan en una nueva `Tuple`.
        - Esta `Tuple` se añade a la lista `notas`.
    - El `Timer1` debe estar habilitado al inicio (por ejemplo, en el `Form_Load`).
    - En cada disparo del `Timer1_Tick` (cada segundo):
        - Se itera sobre la lista `notas` (`For Each nota In notas`). Se utiliza una copia temporal de la lista para evitar errores si se modifican elementos durante la iteración.
        - Se comprueba si la hora actual (`DateTime.Now`) es igual o posterior (`>=`) a la fecha y hora del recordatorio de la nota (`nota.Item2`).
        - Si la condición se cumple, se muestra un `MessageBox` con el texto de la nota (`nota.Item1`).
        - Es crucial (aunque no se muestra en la fuente) que, después de mostrar la alarma, esa nota se elimine de la lista `notas` para evitar que la alarma se repita infinitamente.

Programa 119. Aplicación de calendario simple

- **Explicación del Ejercicio:** Este programa utiliza el control **MonthCalendar** para mostrar un calendario y permite al usuario **visualizar la fecha seleccionada** en una etiqueta.
- **Controles Necesarios:**
    - **MonthCalendar:**
        - **Nombre (Name):** `MonthCalendar1`. Identificador del control de calendario.
        - **Propiedad `SelectionStart`:** Devuelve la fecha de inicio del rango de fechas seleccionado. Para una selección de una sola fecha, esta es la fecha seleccionada.
        - **Evento `DateChanged`:** `MonthCalendar1_DateChanged`. Este evento se dispara cada vez que el usuario selecciona una nueva fecha en el calendario. Contiene la lógica para actualizar la etiqueta con la fecha elegida.
        - **Utilización:** Proporciona una interfaz visual de calendario para que el usuario navegue y seleccione fechas.
    - **Label:**
        - **Nombre (Name):** `lblFecha`. Identificador de la etiqueta.
        - **Propiedad `Text`:** Se actualiza para mostrar la fecha seleccionada por el usuario, formateada como una cadena corta (ej., "Fecha seleccionada: DD/MM/AAAA").

        - **Utilización:** Muestra al usuario la fecha que ha elegido en el calendario.
- **Explicación de la Lógica:**
    - El evento `MonthCalendar1_DateChanged` se activa cada vez que el usuario selecciona una fecha diferente en el `MonthCalendar`.
    - El objeto `e` (de tipo `DateRangeEventArgs`) en el manejador del evento contiene información sobre el rango de fechas seleccionado. `e.Start` devuelve la fecha de inicio de este rango, que en una selección simple es la fecha elegida.
    - `e.Start.ToShortDateString()` convierte el objeto `Date` a una cadena de texto en un formato de fecha corto y legible, específico de la configuración regional del sistema.
    - `lblFecha.Text` se actualiza para mostrar la cadena "Fecha seleccionada: " seguida de la fecha formateada.

Programa 120. Agenda con calendario integrado

- **Explicación del Ejercicio:** Este programa combina un control de calendario (`MonthCalendar`) con una funcionalidad de notas. Permite al usuario **agregar notas a fechas específicas y visualizarlas al seleccionar una fecha** en el calendario.
- **Controles Necesarios:**
    - **MonthCalendar:**
        - **Nombre (Name):** `mcAgenda`. Identificador del control de calendario.
        - **Propiedad `SelectionStart`:** Devuelve la fecha seleccionada actualmente en el calendario. Se utiliza para asociar las notas a una fecha específica y para actualizar la lista de notas cuando cambia la selección.
        - **Evento `DateChanged`:** `mcAgenda_DateChanged`. Se dispara cuando el usuario selecciona una nueva fecha. Llama al método `ActualizarLista` con la fecha seleccionada.
        - **Utilización:** Proporciona una interfaz visual para la navegación y selección de fechas, que son claves para la organización de las notas.
    - **ListBox:**
        - **Nombre (Name):** `lstNotas`. Control que muestra las notas asociadas a la fecha seleccionada.
        - **Colección `Items`:** Almacena y visualiza las cadenas de texto de las notas.
        - **Método `Items.Clear()`:** Limpia la lista de notas visible antes de cargar las de la nueva fecha.
        - **Método `Items.Add(n)`:** Añade una nota (`n`) a la lista visible.
        - **Utilización:** Muestra las notas correspondientes a la fecha actualmente seleccionada en el calendario.
    - **TextBox:**
        - **Nombre (Name):** `txtNota`. Campo de entrada para escribir el texto de la nota.
        - **Propiedad `Text`:** Contiene el texto de la nota que el usuario desea agregar.
        - **Utilización:** Permite al usuario introducir el contenido de una nueva nota.
    - **Button:**
        - **Nombre (Name):** `btnAgregarNota`. Identificador del botón.

        - **Texto (Text):** "Agregar Nota". Texto visible en el botón.
        - **Evento `Click`:** `btnAgregarNota_Click`. Se activa al hacer clic. Contiene la lógica para guardar la nota en el diccionario de notas y actualizar la lista.
        - **Utilización:** Dispara la acción de guardar una nota para la fecha seleccionada.
- **Variables y Clases de Soporte (No controles visuales):**
    - **`notas` (Dictionary(Of Date, List(Of String))):** Un diccionario a nivel de clase que es la estructura de datos central. Cada **clave** es un objeto `Date` (representando una fecha sin hora) y cada **valor** es una `List(Of String)` que almacena todas las notas de texto asociadas a esa fecha.
- **Explicación de la Lógica:**
    - El diccionario `notas` se declara a nivel de clase para almacenar las notas, mapeando fechas a listas de cadenas.
    - En el evento `btnAgregarNota_Click`:
        - `Dim fecha As Date = mcAgenda.SelectionStart` obtiene la fecha seleccionada en el `MonthCalendar`.
        - `If Not notas.ContainsKey(fecha) Then notas(fecha) = New List(Of String)`: Si la fecha seleccionada aún no tiene entradas en el diccionario `notas`, se crea una nueva lista vacía de cadenas para esa fecha.
        - `notas(fecha).Add(txtNota.Text)` añade el texto de `txtNota` a la lista de notas correspondiente a la fecha seleccionada.
        - `ActualizarLista(fecha)` se llama para refrescar `lstNotas` y mostrar la nota recién añadida.
    - El evento `mcAgenda_DateChanged` se dispara cuando el usuario selecciona otra fecha. Simplemente llama a `ActualizarLista(e.Start)` para cargar las notas de la nueva fecha.
    - El método `ActualizarLista(fecha As Date)`:
        - `lstNotas.Items.Clear()` limpia `lstNotas`.
        - `If notas.ContainsKey(fecha) Then` verifica si hay notas almacenadas para la fecha dada.
        - Si hay notas, un bucle `For Each n In notas(fecha)` itera sobre ellas, añadiendo cada nota (`n`) a `lstNotas` para que sean visibles.

Programa 121. Agenda con alarma

- **Explicación del Ejercicio:** Este programa permite al usuario **agregar notas con una fecha y hora específicas de recordatorio**. Cuando el momento de la alarma llega, la aplicación muestra un `MessageBox` con el texto de la nota.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtNota`. Este es el identificador único del control en el código.

        - **Propiedad `Text`:** Contiene el texto de la nota que el usuario desea guardar y que aparecerá en el recordatorio.
        - **Utilización:** Permite al usuario escribir el contenido de la nota para la alarma.
    - **Button:**
        - **Nombre (Name):** `btnAgregar`. Identificador del botón.
        - **Texto (Text):** "Agregar". Define el texto visible en la superficie del botón.
        - **Evento `Click`:** `btnAgregar_Click`. Se activa cuando el usuario pulsa el botón. Contiene la lógica para añadir la nota y su fecha/hora de recordatorio a una lista y al `ListBox`.
        - **Utilización:** El botón sirve como el disparador para guardar la nota y configurar la alarma.
    - **DateTimePicker:**
        - **Nombre (Name):** `dtpAlarma`. Control que permite al usuario seleccionar una fecha y hora.
        - **Propiedad `Value`:** Contiene la fecha y hora seleccionadas por el usuario. Esta es la fecha y hora en que se disparará la alarma.
        - **Utilización:** Proporciona una interfaz amigable para que el usuario establezca el momento exacto del recordatorio.
    - **ListBox:**
        - **Nombre (Name):** `lstAlarmas`. Control para mostrar las alarmas programadas.
        - **Colección `Items`:** Contiene los elementos (cadenas de texto) que representan cada alarma con su nota y fecha/hora.
        - **Método `Items.Add()`:** Añade la cadena que representa la alarma a la lista visible.
        - **Utilización:** Muestra al usuario un resumen de las alarmas que ha configurado.
    - **Timer:**
        - **Nombre (Name):** `Timer1`. Componente no visible que genera eventos en intervalos regulares.
        - **Propiedad `Interval`:** 1000 (milisegundos). Se configura para disparar su evento `Tick` cada 1000 milisegundos (1 segundo).
        - **Evento `Tick`:** `Timer1_Tick`. En cada disparo, itera sobre la lista de alarmas guardadas para comprobar si alguna fecha/hora de recordatorio se ha cumplido.
        - **Utilización:** Es el motor del programa, verificando continuamente si alguna alarma debe dispararse.
- **Explicación de la Lógica:**
    - Se declara una lista `alarmas` de tipo `List(Of Tuple(Of String, DateTime))` a nivel de clase para almacenar las notas y sus respectivas fechas/horas de alarma.
    - Cuando el usuario hace clic en `btnAgregar`, el texto de `txtNota` y el valor de `dtpAlarma.Value` se añaden como una nueva tupla a la lista `alarmas`. El contacto también se muestra en `lstAlarmas`.

- El `Timer1` se dispara cada segundo (debido a `Interval: 1000`). En su evento `Timer1_Tick`, se itera sobre la lista `alarmas`.
- Para cada alarma, se compara la fecha y hora actual (`DateTime.Now`) con la fecha y hora de la alarma (`alarma.Item2`). Si la hora actual es igual o posterior, se muestra un `MessageBox` con el mensaje de la alarma (`alarma.Item1`) y esa alarma se elimina de la lista para evitar repeticiones.

Programa 122. Agenda con notificaciones

- **Explicación del Ejercicio:** Este programa gestiona una lista de notas y muestra una **notificación periódica** sobre la cantidad de notas pendientes.
- **Controles Necesarios:**
  - **ListBox:**
    - **Nombre (Name):** `lstNotas`. Control que muestra la lista de notas.
    - **Colección `Items`:** Contiene los elementos (cadenas de texto) que representan cada nota.
    - **Método `Items.Add()`:** Añade el texto de la nota a la lista visible.
    - **Utilización:** Muestra visualmente las notas que el usuario ha agregado.
  - **Button:**
    - **Nombre (Name):** `btnAgregar`. Identificador del botón.
    - **Texto (Text):** "Agregar Nota". Texto visible en el botón.
    - **Evento `Click`:** `btnAgregar_Click`. Se activa al hacer clic. Contiene la lógica para añadir la nota a la lista interna y al `ListBox`.
    - **Utilización:** Permite al usuario añadir nuevas notas a la agenda.
  - **TextBox:**
    - **Nombre (Name):** `txtNota`. Campo de entrada para el texto de la nota.
    - **Propiedad `Text`:** Contiene el texto que el usuario desea añadir como nota.
    - **Utilización:** Es el campo donde el usuario escribe el contenido de la nota.
  - **Timer:**
    - **Nombre (Name):** `Timer1`. Componente no visible que genera eventos en intervalos regulares.
    - **Propiedad `Interval`:** 60000 (1 minuto). Se configura para disparar su evento `Tick` cada 60000 milisegundos (1 minuto).
    - **Evento `Tick`:** `Timer1_Tick`. En cada disparo, comprueba si hay notas pendientes y muestra una notificación.
    - **Utilización:** Es el motor que desencadena las notificaciones periódicas.
- **Explicación de la Lógica:**
  - Se utiliza una lista `notas` de tipo `List(Of String)` a nivel de clase para almacenar las notas.
  - Cuando el usuario hace clic en `btnAgregar`, el texto de `txtNota` se añade tanto a la lista `notas` como al `lstNotas` visible.
  - El `Timer1` se dispara cada 60 segundos. En su evento `Timer1_Tick`, se verifica si la lista `notas` contiene elementos (`notas.Count > 0`).
  - Si hay notas, se muestra un `MessageBox` informando al usuario sobre la cantidad de notas pendientes.

Programa 123. Agenda con múltiples formularios

- **Explicación del Ejercicio:** Este programa demuestra cómo una aplicación puede tener **múltiples ventanas o formularios**, permitiendo que un formulario principal (`Form1`) pueda **abrir y mostrar otra ventana secundaria** (`Form2`).
- **Controles Necesarios:**
    - **Button (en `Form1`):**
        - **Nombre (Name):** `btnAbrirForm2`. Identificador del botón.
        - **Texto (Text):** "Abrir Formulario 2". Texto visible en el botón que indica su función.
        - **Evento `Click`:** `btnAbrirForm2_Click`. Se dispara cuando el usuario hace clic en el botón. Contiene la lógica para crear y mostrar una nueva instancia de `Form2`.
        - **Utilización:** Actúa como el disparador para abrir la ventana secundaria.
    - **Formulario `Form2` (Formulario Adicional):**
        - **Requisito Adicional:** Se debe crear un **segundo formulario** en el proyecto de Visual Basic .NET con el nombre `Form2`. Puede estar vacío o contener un `Label` u otro control simple.
        - **Utilización:** Es la ventana que se abre cuando el usuario interactúa con `Form1`, demostrando la capacidad de la aplicación para gestionar múltiples interfaces.
- **Explicación de la Lógica:**
    - Al hacer clic en `btnAbrirForm2`, se ejecuta el evento `btnAbrirForm2_Click`.
    - `Dim f2 As New Form2()` crea una **nueva instancia** de la clase `Form2`. Esto significa que se crea un nuevo objeto `Form2` en la memoria.
    - `f2.Show()` hace que esta nueva instancia del formulario (`f2`) sea **visible en la pantalla**, abriendo una nueva ventana independiente de `Form1`.

Programa 124. Agenda con login de usuario

- **Explicación del Ejercicio:** Este programa simula un **proceso de inicio de sesión** básico, donde el usuario debe introducir un nombre de usuario y una contraseña predefinidos para acceder a la aplicación.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtUsuario`. Campo de entrada para el nombre de usuario.
        - **Propiedad `Text`:** Contiene el nombre de usuario ingresado por el usuario, que se compara con el usuario predefinido ("admin").
        - **Utilización:** Permite al usuario introducir su nombre de usuario.
    - **TextBox:**
        - **Nombre (Name):** `txtPassword`. Campo de entrada para la contraseña.
        - **Propiedad `PasswordChar`:** `*`. Esta propiedad es importante, ya que hace que los caracteres ingresados en el `TextBox` se muestren como asteriscos

(*) en lugar de los caracteres reales, ocultando la contraseña para mayor seguridad visual.
        - **Propiedad `Text`:** Contiene la contraseña ingresada por el usuario, que se compara con la contraseña predefinida ("1234").
        - **Utilización:** Permite al usuario introducir su contraseña de forma segura.
    - **Button:**
        - **Nombre (Name):** `btnLogin`. Identificador del botón.
        - **Texto (Text):** "Iniciar Sesión". Texto visible en el botón.
        - **Evento `Click`:** `btnLogin_Click`. Se activa al hacer clic. Contiene la lógica para verificar las credenciales.
        - **Utilización:** Dispara la acción de inicio de sesión.
    - **Label:**
        - **Nombre (Name):** `lblMensaje`. Identificador de la etiqueta.
        - **Propiedad `Text`:** Muestra mensajes al usuario como "Acceso concedido" o "Usuario o contraseña incorrecta", dependiendo del resultado de la validación.
        - **Utilización:** Informa al usuario sobre el éxito o fracaso del intento de inicio de sesión.
- **Explicación de la Lógica:**
    - Cuando el usuario hace clic en `btnLogin`, el evento `btnLogin_Click` se ejecuta.
    - El programa compara el texto de `txtUsuario` con la cadena "admin" y el texto de `txtPassword` con la cadena "1234".
    - Si ambas condiciones son verdaderas (coinciden usuario y contraseña), `lblMensaje.Text` se actualiza a "Acceso concedido". De lo contrario, `lblMensaje.Text` muestra "Usuario o contraseña incorrecta".

Programa 125. Agenda con niveles de usuario

- **Explicación del Ejercicio:** Este programa demuestra cómo controlar el **acceso a ciertas funcionalidades o roles** en la aplicación según el nivel de usuario seleccionado de un `ComboBox`.
- **Controles Necesarios:**
    - **ComboBox:**
        - **Nombre (Name):** `cmbNivel`. Control que presenta una lista desplegable de opciones de nivel de usuario.
        - **Colección `Items`:** Se carga con las opciones "Usuario" y "Administrador" en el evento `Form1_Load`.
        - **Propiedad `SelectedItem`:** Contiene la opción seleccionada actualmente por el usuario. Se utiliza para verificar el nivel de acceso.
        - **Utilización:** Permite al usuario elegir su nivel de rol en la aplicación.
    - **Button:**
        - **Nombre (Name):** `btnLogin`. Identificador del botón.
        - **Propiedad `Text`:** "Login". Texto visible en el botón.

        - **Evento `Click`:** `btnLogin_Click`. Se activa al hacer clic. Contiene la lógica para determinar el mensaje de acceso según el nivel seleccionado.
        - **Utilización:** Simula el inicio de sesión y la verificación de permisos.
    - **Label:**
        - **Nombre (Name):** `lblMensaje`. Identificador de la etiqueta.
        - **Propiedad `Text`:** Muestra un mensaje al usuario indicando el tipo de acceso (ej., "Acceso de administrador" o "Acceso de usuario normal").
        - **Utilización:** Informa al usuario sobre su nivel de acceso concedido.
- **Explicación de la Lógica:**
    - En el evento `Form1_Load`, el `ComboBox cmbNivel` se carga con "Usuario" y "Administrador", y "Usuario" se selecciona por defecto.
    - Al hacer clic en `btnLogin`, el programa verifica el valor de `cmbNivel.SelectedItem`.
    - Si la opción seleccionada es "Administrador", `lblMensaje` muestra "Acceso de administrador". De lo contrario (si es "Usuario" u otra opción), `lblMensaje` muestra "Acceso de usuario normal".

Programa 126. Agenda con encriptación de datos

- **Explicación del Ejercicio:** Este programa demuestra una forma básica de **cifrar texto utilizando la codificación Base64**. Es una técnica común para codificar datos binarios en un formato de cadena ASCII.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtDato`. Campo de entrada para el texto que se desea encriptar.
        - **Propiedad `Text`:** Contiene el texto plano que el usuario introduce y que será convertido a Base64.
        - **Utilización:** Permite al usuario introducir el dato original para ser cifrado.
    - **Button:**
        - **Nombre (Name):** `btnEncriptar`. Identificador del botón.
        - **Texto (Text):** "Encriptar". Texto visible en el botón.
        - **Evento `Click`:** `btnEncriptar_Click`. Se activa al hacer clic. Contiene la lógica para realizar la codificación Base64.
        - **Utilización:** Dispara la acción de encriptación del texto.
    - **Label:**
        - **Nombre (Name):** `lblResultado`. Identificador de la etiqueta.
        - **Propiedad `Text`:** Muestra la cadena de texto resultante después de la codificación Base64.
        - **Utilización:** Presenta el texto codificado al usuario.
- **Importación de Namespace:**
    - **`Imports System.Text`:** Necesario para utilizar la clase `Encoding`.
- **Explicación de la Lógica:**
    - Cuando se hace clic en `btnEncriptar`, el evento `btnEncriptar_Click` se ejecuta.

- `Encoding.UTF8.GetBytes(txtDato.Text)` convierte el texto de `txtDato` en una **matriz de bytes** utilizando la codificación UTF-8.
- `Convert.ToBase64String()` toma esa matriz de bytes y la convierte en una **cadena Base64**, que es la representación "encriptada" (codificada) del texto original. El resultado se muestra en `lblResultado`.

Programa 127. Agenda con desencriptación de datos

- **Explicación del Ejercicio:** Este programa complementa el anterior, demostrando cómo **descodificar una cadena Base64 de vuelta a su texto original**.
- **Controles Necesarios:**
  - **TextBox:**
    - **Nombre (Name):** `txtEncriptado`. Campo de entrada para la cadena Base64 que se desea desencriptar.
    - **Propiedad `Text`:** Contiene la cadena Base64 ingresada por el usuario, la cual será convertida a texto plano.
    - **Utilización:** Permite al usuario introducir el texto codificado en Base64.
  - **Button:**
    - **Nombre (Name):** `btnDesencriptar`. Identificador del botón.
    - **Texto (Text):** "Desencriptar". Texto visible en el botón.
    - **Evento `Click`:** `btnDesencriptar_Click`. Se activa al hacer clic. Contiene la lógica para realizar la decodificación Base64.
    - **Utilización:** Dispara la acción de desencriptación del texto.
  - **Label:**
    - **Nombre (Name):** `lblResultado`. Identificador de la etiqueta.
    - **Propiedad `Text`:** Muestra la cadena de texto resultante después de la decodificación Base64.
    - **Utilización:** Presenta el texto original al usuario.
- **Importación de Namespace:**
  - **`Imports System.Text`:** Necesario para utilizar la clase `Encoding`.
- **Explicación de la Lógica:**
  - Al hacer clic en `btnDesencriptar`, el evento `btnDesencriptar_Click` se ejecuta.
  - `Convert.FromBase64String(txtEncriptado.Text)` convierte la cadena Base64 de `txtEncriptado` de nuevo a una **matriz de bytes**.
  - `Encoding.UTF8.GetString()` toma esa matriz de bytes y la convierte de nuevo a la **cadena de texto original**. El resultado se muestra en `lblResultado`.

Programa 128. Calculadora científica simple

- **Explicación del Ejercicio:** Este programa implementa una **calculadora básica con operaciones de suma, resta, multiplicación y división**, además de la entrada de números.
- **Controles Necesarios:**
  - **TextBox:**
    - **Nombre (Name):** `txtPantalla`. Actúa como la pantalla de la calculadora donde se muestran los números y resultados.

        - **Propiedad `Text`:** Contiene los dígitos ingresados por el usuario o el resultado de una operación. Se le concatenan los dígitos.
        - **Método `Clear()`:** Se utiliza para limpiar la pantalla después de seleccionar una operación, preparando la entrada del segundo número.
        - **Utilización:** Es la interfaz principal para mostrar la interacción numérica de la calculadora.
    - **Buttons (para números del 0 al 9):**
        - **Nombres (Names):** `btn0` a `btn9`. Se asume que hay un botón por cada dígito.
        - **Evento `Click`:** `btnNumero_Click` (manejado por todos los botones numéricos). Este evento añade el dígito del botón pulsado a `txtPantalla.Text`.
        - **Utilización:** Permiten al usuario ingresar los números para las operaciones.
    - **Buttons (para operaciones):**
        - **Nombres (Names):** `btnSumar`, `btnRestar`, `btnMultiplicar`, `btnDividir`.
        - **Texto (Text):** "+", "-", "*", "/". Los símbolos de operación.
        - **Evento `Click`:** `btnOperacion_Click` (manejado por los botones de operación). Guarda el número actual de `txtPantalla` en `valor1`, el símbolo de la operación en `operacion`, y limpia `txtPantalla`.
        - **Utilización:** Permiten al usuario seleccionar la operación aritmética a realizar.
    - **Button:**
        - **Nombre (Name):** `btnIgual`.
        - **Texto (Text):** "=".
        - **Evento `Click`:** `btnIgual_Click`. Calcula el resultado final basándose en la `operacion` y los valores `valor1` y `valor2`.
        - **Utilización:** Desencadena el cálculo y muestra el resultado.
- **Variables de Soporte (No controles visuales):**
    - **`operacion` (String):** Variable a nivel de clase que almacena el símbolo de la operación matemática que el usuario ha seleccionado (ej., "+", "-", "*", "/").
    - **`valor1` (Double):** Variable a nivel de clase que almacena el primer operando de la operación.
- **Explicación de la Lógica:**
    - Las variables `operacion` y `valor1` se declaran a nivel de clase para mantener el estado de la calculadora entre diferentes clics.
    - Los eventos `btnNumero_Click` (compartidos por todos los botones numéricos) concatenan el dígito presionado a `txtPantalla.Text`.
    - Cuando un botón de operación es clicado (`btnOperacion_Click`), el valor actual en `txtPantalla` se convierte a `Double` con `Val()` y se guarda en `valor1`. El `Text` del botón de operación (ej., "+") se guarda en `operacion`, y `txtPantalla` se limpia para la entrada del segundo número.
    - Al hacer clic en `btnIgual`, el valor actual de `txtPantalla` se toma como `valor2`. Un `Select Case` evalúa la `operacion` almacenada y realiza el

cálculo correspondiente entre `valor1` y `valor2`. El resultado se muestra en `txtPantalla.Text`.

Programa 129. Calculadora con funciones trigonométricas

- **Explicación del Ejercicio:** Este programa extiende la calculadora básica para incluir **funciones trigonométricas** como seno (Sin), coseno (Cos) y tangente (Tan).
- **Controles Necesarios:**
  - **TextBox:**
    - **Nombre (Name):** `txtPantalla`. Actúa como la pantalla de la calculadora.
    - **Propiedad `Text`:** Contiene el número al que se aplicará la función trigonométrica. Se actualiza con el resultado de la operación.
    - **Utilización:** Muestra el número de entrada y el resultado trigonométrico.
  - **Button:**
    - **Nombre (Name):** `btnSin`.
    - **Texto (Text):** "Sin". Texto visible en el botón.
    - **Evento `Click`:** `btnSin_Click`. Aplica la función `Math.Sin()` al valor en `txtPantalla`.
    - **Utilización:** Permite al usuario calcular el seno de un número.
  - **Button:**
    - **Nombre (Name):** `btnCos`.
    - **Texto (Text):** "Cos". Texto visible en el botón.
    - **Evento `Click`:** `btnCos_Click`. Aplica la función `Math.Cos()` al valor en `txtPantalla`.
    - **Utilización:** Permite al usuario calcular el coseno de un número.
  - **Button:**
    - **Nombre (Name):** `btnTan`.
    - **Texto (Text):** "Tan". Texto visible en el botón.
    - **Evento `Click`:** `btnTan_Click`. Aplica la función `Math.Tan()` al valor en `txtPantalla`.
    - **Utilización:** Permite al usuario calcular la tangente de un número.
- **Explicación de la Lógica:**
  - Cada botón trigonométrico tiene su propio evento `_Click` asociado.
  - Cuando se presiona uno de estos botones, el valor numérico actual en `txtPantalla` se convierte a `Double` con `Val()`.
  - Luego, se aplica la función trigonométrica correspondiente (`Math.Sin()`, `Math.Cos()`, o `Math.Tan()`) a este valor.
  - El resultado de la función trigonométrica se convierte de nuevo a una cadena y reemplaza el texto en `txtPantalla`.
  - **Nota:** Las funciones `Math.Sin`, `Math.Cos`, `Math.Tan` esperan argumentos en **radianes**. Si el usuario ingresa grados, sería necesaria una conversión de grados a radianes antes de aplicar la función (por ejemplo, `valor * Math.PI / 180`).

Programa 130. Calculadora con funciones logarítmicas

- **Explicación del Ejercicio:** Este programa añade **funciones logarítmicas** a la calculadora: logaritmo base 10 (Log) y logaritmo natural (Ln).
- **Controles Necesarios:**
  - **TextBox:**
    - **Nombre (Name):** `txtPantalla`. La pantalla de la calculadora.
    - **Propiedad `Text`:** Contiene el número al que se aplicará la función logarítmica. Se actualiza con el resultado.
    - **Utilización:** Muestra el número de entrada y el resultado logarítmico.
  - **Button:**
    - **Nombre (Name):** `btnLog`.
    - **Texto (Text):** "Log". Texto visible en el botón.
    - **Evento `Click`:** `btnLog_Click`. Aplica la función `Math.Log10()` al valor en `txtPantalla`.
    - **Utilización:** Permite al usuario calcular el logaritmo base 10 de un número.
  - **Button:**
    - **Nombre (Name):** `btnLn`.
    - **Texto (Text):** "Ln". Texto visible en el botón.
    - **Evento `Click`:** `btnLn_Click`. Aplica la función `Math.Log()` (logaritmo natural) al valor en `txtPantalla`.
    - **Utilización:** Permite al usuario calcular el logaritmo natural de un número.
- **Explicación de la Lógica:**
  - Cada botón logarítmico tiene un evento `_Click` asociado.
  - Al hacer clic en `btnLog`, `Math.Log10(Val(txtPantalla.Text))` calcula el **logaritmo en base 10** del número que se encuentra en `txtPantalla`, y el resultado se muestra directamente en `txtPantalla.Text`.
  - Al hacer clic en `btnLn`, `Math.Log(Val(txtPantalla.Text))` calcula el **logaritmo natural** (base e) del número en `txtPantalla`, y el resultado se muestra en `txtPantalla.Text`.

Programa 131. Calculadora con potencias y raíces

- **Explicación del Ejercicio:** Este programa extiende la calculadora para incluir operaciones de **potencia (^) y raíz cuadrada (√)**.
- **Controles Necesarios:**
  - **TextBox:**
    - **Nombre (Name):** `txtPantalla`. La pantalla de la calculadora.
    - **Propiedad `Text`:** Muestra los números de entrada, el resultado intermedio para la potencia o el resultado final.
    - **Método `Clear()`:** Se utiliza para limpiar la pantalla después de seleccionar la operación de potencia, para la entrada del segundo operando.
    - **Utilización:** Es la interfaz para mostrar las entradas y resultados.
  - **Button:**
    - **Nombre (Name):** `btnPotencia`.
    - **Texto (Text):** "^". Símbolo para la operación de potencia.

        - **Evento `Click`:** `btnPotencia_Click`. Guarda el valor actual de `txtPantalla` en `valor1`, establece la variable `operacion` a "^", y limpia `txtPantalla` para la entrada del exponente.
        - **Utilización:** Permite al usuario iniciar una operación de potencia.
    - **Button:**
        - **Nombre (Name):** `btnRaiz`.
        - **Texto (Text):** "√". Símbolo para la operación de raíz cuadrada.
        - **Evento `Click`:** `btnRaiz_Click`. Calcula la raíz cuadrada del número en `txtPantalla` y muestra el resultado directamente.
        - **Utilización:** Permite al usuario calcular la raíz cuadrada de un número.
    - **Button:**
        - **Nombre (Name):** `btnIgual`.
        - **Texto (Text):** "=".
        - **Evento `Click`:** `btnIgual_Click`. Si la operación pendiente es potencia, usa `Math.Pow()` para calcular el resultado.
        - **Utilización:** Desencadena el cálculo de la potencia.
- **Variables de Soporte (No controles visuales):**
    - **`valor1` (Double):** Almacena el primer operando (la base en la potencia).
    - **`operacion` (String):** Almacena la operación pendiente (en este caso, "^").
- **Explicación de la Lógica:**
    - Las variables `valor1` y `operacion` se utilizan para almacenar el estado de la operación de potencia.
    - Cuando se hace clic en `btnPotencia`, el valor actual de `txtPantalla` se guarda en `valor1`, la variable `operacion` se establece en "^", y `txtPantalla` se borra para que el usuario ingrese el exponente.
    - Al hacer clic en `btnRaiz`, `Math.Sqrt(Val(txtPantalla.Text))` calcula la raíz cuadrada del valor actual en `txtPantalla`, y el resultado se muestra directamente en `txtPantalla.Text`.
    - Cuando se presiona `btnIgual` (asumiendo que los botones numéricos y la lógica de la calculadora básica están presentes), si la `operacion` pendiente es "^", `Math.Pow(valor1, valor2)` (donde `valor2` es el valor en `txtPantalla` en ese momento) calcula la potencia, mostrando el resultado en `txtPantalla.Text`.

Programa 132. Calculadora con historial de operaciones

- **Explicación del Ejercicio:** Este programa implementa una calculadora que guarda un **historial de las operaciones realizadas** en un `ListBox`.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtPantalla`. La pantalla de la calculadora.
        - **Propiedad `Text`:** Muestra los números y los resultados.
        - **Método `Clear()`:** Limpia la pantalla después de una operación.
        - **Utilización:** Interfaz para la entrada y visualización de resultados.
    - **ListBox:**

        - **Nombre (Name):** `lstHistorial`. Control donde se muestra el historial de las operaciones.
        - **Colección `Items`:** Almacena y visualiza cada operación completa con su resultado.
        - **Método `Items.Add()`:** Añade una cadena de texto que describe la operación y su resultado al historial.
        - **Utilización:** Proporciona un registro visual de todas las operaciones realizadas por el usuario.
    - **Buttons (numéricos y de operaciones):**
        - Se asume la presencia de botones para los números (0-9) y operaciones (+, -, *, /), similares a los del Programa 128.
        - **Evento `Click` de Operaciones (`btnOperacion_Click`):** Almacena el primer valor (`valor1`) y la operación (`operacion`).
        - **Utilización:** Permiten la interacción básica de la calculadora.
    - **Button:**
        - **Nombre (Name):** `btnIgual`.
        - **Texto (Text):** "=".
        - **Evento `Click`:** `btnIgual_Click`. Calcula el resultado y lo añade al historial.
        - **Utilización:** Desencadena el cálculo y el registro en el historial.
- **Variables de Soporte (No controles visuales):**
    - **`valor1` (Double):** Almacena el primer operando.
    - **`operacion` (String):** Almacena la operación pendiente.
- **Explicación de la Lógica:**
    - Las variables `valor1` y `operacion` se utilizan de manera similar a una calculadora básica.
    - El evento `btnOperacion_Click` (asociado a los botones +, -, *, /) almacena el primer valor numérico de `txtPantalla` en `valor1`, el símbolo de la operación en `operacion`, y luego limpia `txtPantalla`.
    - Cuando se presiona `btnIgual`, el valor actual de `txtPantalla` se obtiene como `valor2`. Se realiza la operación almacenada en `operacion` entre `valor1` y `valor2`, y el `resultado` se muestra en `txtPantalla`.
    - Crucialmente, se añade una cadena que describe la operación completa (`valor1 & " " & operacion & " " & valor2 & " = " & resultado`) a `lstHistorial.Items`, registrando así la operación.

Programa 133. Calculadora con memoria de resultados

- **Explicación del Ejercicio:** Este programa añade **funciones de memoria** a la calculadora, permitiendo al usuario guardar, recuperar y borrar un valor en memoria.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtPantalla`. La pantalla principal de la calculadora.
        - **Propiedad `Text`:** Muestra el valor actual para guardar, o el valor recuperado de la memoria.

        - **Utilización:** Sirve como la interfaz para la interacción numérica.
    - **Button:**
        - **Nombre (Name):** `btnMS`.
        - **Texto (Text):** "MS" (Memory Store).
        - **Evento `Click`:** `btnMS_Click`. Guarda el valor actual de `txtPantalla` en la variable `memoria`.
        - **Utilización:** Permite al usuario almacenar el valor que se muestra en la pantalla en la memoria.
    - **Button:**
        - **Nombre (Name):** `btnMR`.
        - **Texto (Text):** "MR" (Memory Recall).
        - **Evento `Click`:** `btnMR_Click`. Recupera el valor de la variable `memoria` y lo muestra en `txtPantalla`.
        - **Utilización:** Permite al usuario recuperar y mostrar el valor guardado en la memoria en la pantalla.
    - **Button:**
        - **Nombre (Name):** `btnMC`.
        - **Texto (Text):** "MC" (Memory Clear).
        - **Evento `Click`:** `btnMC_Click`. Restablece la variable `memoria` a 0.
        - **Utilización:** Permite al usuario borrar el valor almacenado en la memoria.
- **Variables de Soporte (No controles visuales):**
    - **`memoria` (Double):** Variable a nivel de clase, inicializada a 0, que almacena el valor numérico guardado en la memoria de la calculadora.
- **Explicación de la Lógica:**
    - La variable `memoria` se declara a nivel de clase para persistir su valor entre las operaciones.
    - Cuando se hace clic en `btnMS`, el valor numérico actual en `txtPantalla` (obtenido con `Val()`) se asigna a la variable `memoria`.
    - Al hacer clic en `btnMR`, el valor almacenado en `memoria` se convierte a cadena y se asigna a `txtPantalla.Text`, mostrando el valor recordado.
    - Cuando se hace clic en `btnMC`, la variable `memoria` se reinicia a 0, borrando el contenido de la memoria.

Programa 134. Juego de preguntas con puntuación

- **Explicación del Ejercicio:** Este programa implementa un **juego de preguntas y respuestas** que lleva un seguimiento de la **puntuación del jugador**. El juego avanza a la siguiente pregunta si la respuesta es correcta o incorrecta, y al final muestra el puntaje total.
- **Controles Necesarios:**
    - **Label:**
        - **Nombre (Name):** `lblPregunta`. Identificador de la etiqueta que muestra la pregunta actual.
        - **Propiedad `Text`:** Se actualiza con el texto de la pregunta actual (`preguntas(indice)`) o con el mensaje de fin de juego.
        - **Utilización:** Muestra al usuario la pregunta a responder.

    - **TextBox:**
        - **Nombre (Name):** `txtRespuesta`. Campo de entrada donde el usuario escribe su respuesta.
        - **Propiedad `Text`:** Contiene la respuesta ingresada por el usuario, que se compara con la respuesta correcta.
        - **Método `Clear()`:** Se utiliza para vaciar el `TextBox` después de cada respuesta, preparándolo para la siguiente pregunta.
        - **Utilización:** Permite al usuario introducir su respuesta.
    - **Button:**
        - **Nombre (Name):** `btnVerificar`. Identificador del botón.
        - **Texto (Text):** "Verificar". Texto visible en el botón.
        - **Evento `Click`:** `btnVerificar_Click`. Se activa al hacer clic. Contiene la lógica para verificar la respuesta, actualizar el puntaje, avanzar a la siguiente pregunta y manejar el fin del juego.
        - **Utilización:** Dispara la verificación de la respuesta y el avance del juego.
    - **Label:**
        - **Nombre (Name):** `lblPuntaje`. Identificador de la etiqueta que muestra la puntuación actual.
        - **Propiedad `Text`:** Se actualiza para mostrar el puntaje del jugador (ej., "Puntaje: 1").
        - **Utilización:** Informa al usuario sobre su rendimiento en el juego.
- **Variables de Soporte (No controles visuales):**
    - **`preguntas` (List(Of String)):** Una lista a nivel de clase que almacena el texto de todas las preguntas del juego.
    - **`respuestas` (List(Of String)):** Una lista a nivel de clase que almacena las respuestas correctas correspondientes a cada pregunta.
    - **`indice` (Integer):** Variable a nivel de clase que rastrea la posición de la pregunta actual en las listas `preguntas` y `respuestas`.
    - **`puntaje` (Integer):** Variable a nivel de clase que almacena la puntuación acumulada del jugador.
- **Explicación de la Lógica:**
    - En el evento `Form1_Load`, `lblPregunta` se inicializa con la primera pregunta de la lista `preguntas`, y `lblPuntaje` muestra el puntaje inicial (0).
    - Cuando se hace clic en `btnVerificar`, el texto de `txtRespuesta` se compara (ignorando mayúsculas/minúsculas con `.ToLower()`) con la respuesta correcta almacenada en `respuestas(indice)`.
    - Si la respuesta es correcta, la variable `puntaje` se incrementa en 1.
    - La variable `indice` se incrementa para pasar a la siguiente pregunta.
    - Se verifica si aún quedan preguntas por responder (`If indice < preguntas.Count Then`). Si es así, `lblPregunta` se actualiza con la siguiente pregunta y `txtRespuesta` se limpia. Si no quedan preguntas, se muestra un `MessageBox` con el puntaje final y se marca el juego como terminado.
    - `lblPuntaje.Text` se actualiza para mostrar el puntaje actual después de cada verificación.

Programa 135. Juego de preguntas con tiempo límite

- **Explicación del Ejercicio:** Este programa implementa un juego de preguntas y respuestas donde el usuario tiene un **tiempo límite para responder cada pregunta**. Si el tiempo se agota antes de responder, se considera una respuesta incorrecta.
- **Controles Necesarios:**
    - **Label:**
        - **Nombre (Name):** `lblPregunta`. Muestra la pregunta actual.
        - **Propiedad `Text`:** Se actualiza con la pregunta o el mensaje de fin de juego.
        - **Utilización:** Muestra la pregunta al usuario.
    - **TextBox:**
        - **Nombre (Name):** `txtRespuesta`. Campo de entrada para la respuesta del usuario.
        - **Propiedad `Text`:** Contiene la respuesta ingresada por el usuario.
        - **Utilización:** Permite al usuario introducir su respuesta.
    - **Button:**
        - **Nombre (Name):** `btnVerificar`.
        - **Texto (Text):** "Verificar".
        - **Evento `Click`:** `btnVerificar_Click`. Verifica la respuesta y maneja el avance del juego, deteniendo y reiniciando el temporizador.
        - **Utilización:** Dispara la verificación de la respuesta.
    - **Label:**
        - **Nombre (Name):** `lblTiempo`. Muestra el tiempo restante para la pregunta actual.
        - **Propiedad `Text`:** Se actualiza cada segundo para mostrar la cuenta regresiva.
        - **Utilización:** Informa al usuario sobre el tiempo disponible para responder.
    - **Timer:**
        - **Nombre (Name):** `Timer1`. Componente no visible.
        - **Propiedad `Interval`:** 1000. Se configura para disparar su evento `Tick` cada 1000 milisegundos (1 segundo).
        - **Método `Start()`:** Inicia el temporizador.
        - **Método `Stop()`:** Detiene el temporizador.
        - **Evento `Tick`:** `Timer1_Tick`. Decrementa el tiempo restante y maneja el caso en que el tiempo llega a cero.
        - **Utilización:** Controla el tiempo límite para cada pregunta.
- **Variables de Soporte (No controles visuales):**
    - **`tiempoRestante` (Integer):** Almacena los segundos que quedan para responder la pregunta actual.
    - **`preguntas` (List(Of String)):** Lista de preguntas.
    - **`respuestas` (List(Of String)):** Lista de respuestas correctas.
    - **`indicePregunta` (Integer):** Rastrea la pregunta actual.
- **Explicación de la Lógica:**
    - En `Form1_Load`, se inicializa la primera pregunta en `lblPregunta` y se inicia el temporizador con `Timer1.Start()` y `tiempoRestante` establecido.

- En cada `Timer1_Tick` (cada segundo):
    - `tiempoRestante` se decrementa.
    - `lblTiempo.Text` se actualiza para mostrar el tiempo restante.
    - Si `tiempoRestante` llega a 0, el `Timer1` se detiene, se muestra un `MessageBox` indicando que el tiempo ha terminado, y se avanza a la siguiente pregunta (similar a responder incorrectamente).
- Cuando se hace clic en `btnVerificar`:
    - El `Timer1` se detiene.
    - La respuesta del usuario se compara con la respuesta correcta.
    - Se muestra un mensaje de "Correcto" o "Incorrecto".
    - Se avanza a la siguiente pregunta.
    - Si hay más preguntas, `tiempoRestante` se reinicia y el `Timer1` se inicia de nuevo. Si no, el juego termina.

Programa 136. Juego de preguntas con niveles de dificultad

- **Explicación del Ejercicio:** Este programa implementa un juego de preguntas donde el usuario puede **seleccionar un nivel de dificultad** (`Fácil`, `Medio`, `Difícil`) a través de un `ComboBox`, lo que influirá en las preguntas que se muestran.
- **Controles Necesarios:**
    - **Label:**
        - **Nombre (Name):** `lblPregunta`. Identificador de la etiqueta que muestra la pregunta actual.
        - **Propiedad `Text`:** Se actualiza con la pregunta correspondiente al nivel de dificultad seleccionado.
        - **Utilización:** Muestra al usuario la pregunta a responder.
    - **TextBox:**
        - **Nombre (Name):** `txtRespuesta`. Campo de entrada donde el usuario escribe su respuesta.
        - **Propiedad `Text`:** Contiene la respuesta ingresada por el usuario.
        - **Utilización:** Permite al usuario introducir su respuesta.
    - **Button:**
        - **Nombre (Name):** `btnVerificar`. Identificador del botón.
        - **Texto (Text):** "Verificar". Texto visible en el botón.
        - **Utilización:** En una implementación completa, este botón verificaría la respuesta y avanzaría de pregunta.
    - **ComboBox:**
        - **Nombre (Name):** `cmbNivel`. Control que permite al usuario seleccionar el nivel de dificultad.
        - **Colección `Items`:** Se carga con las opciones "Fácil", "Medio", "Difícil".
        - **Propiedad `SelectedItem`:** Contiene el nivel de dificultad seleccionado por el usuario.
        - **Evento `SelectedIndexChanged`:** `cmbNivel_SelectedIndexChanged`. Se dispara cuando el usuario

selecciona un nuevo nivel de la lista. Contiene la lógica para cargar la pregunta inicial del nivel correspondiente.
        - **Utilización:** Permite al usuario elegir la dificultad del juego.
- **Variables de Soporte (No controles visuales):**
    - **`preguntasFacil` (List(Of String)):** Lista de preguntas para el nivel "Fácil".
    - **`respuestasFacil` (List(Of String)):** Lista de respuestas correctas para el nivel "Fácil". (Se asume que existirían listas similares para "Medio" y "Difícil" para un juego completo).
- **Explicación de la Lógica:**
    - Se definen listas separadas de preguntas y respuestas para cada nivel (en el ejemplo, solo `preguntasFacil` y `respuestasFacil`).
    - El evento `cmbNivel_SelectedIndexChanged` se activa cuando el usuario selecciona un nuevo nivel.
    - Si el nivel seleccionado es "Fácil" (`cmbNivel.SelectedItem = "Fácil"`), `lblPregunta.Text` se actualiza con la primera pregunta de la lista `preguntasFacil`.
    - Para una implementación completa, se necesitarían listas de preguntas y respuestas para los niveles "Medio" y "Difícil", y la lógica del botón `btnVerificar` para comprobar la respuesta del nivel actual y avanzar a la siguiente pregunta dentro de ese nivel.

Programa 137. Juego de preguntas con base de datos

- **Explicación del Ejercicio:** Este programa integra una **base de datos SQLite** para almacenar y recuperar preguntas y sus respuestas, lo que permite un juego de preguntas con persistencia de datos.
- **Controles Necesarios:**
    - **Label:**
        - **Nombre (Name):** `lblPregunta`. Identificador de la etiqueta que muestra la pregunta actual.
        - **Propiedad `Text`:** Muestra la pregunta recuperada de la base de datos.
        - **Utilización:** Presenta la pregunta al usuario.
    - **TextBox:**
        - **Nombre (Name):** `txtRespuesta`. Campo de entrada para la respuesta del usuario.
        - **Propiedad `Text`:** Contiene la respuesta ingresada por el usuario, que se compara con la respuesta correcta de la base de datos.
        - **Utilización:** Permite al usuario introducir su respuesta.
    - **Button:**
        - **Nombre (Name):** `btnVerificar`. Identificador del botón.
        - **Texto (Text):** "Verificar". Texto visible en el botón.
        - **Evento `Click`:** `btnVerificar_Click`. Se activa al hacer clic. Contiene la lógica para verificar la respuesta del usuario contra la almacenada en la base de datos.
        - **Utilización:** Dispara la verificación de la respuesta.
- **Requisito Adicional (No es un control visual):**

    - **Librería `System.Data.SQLite`:** Este programa **requiere agregar una referencia** a la librería `System.Data.SQLite`, que generalmente se instala a través del administrador de paquetes NuGet en Visual Studio.
- **Importación de Namespace:**
    - **`Imports System.Data.SQLite`:** Es fundamental para utilizar las clases de conectividad a bases de datos SQLite.
- **Clases y Métodos de SQLite:**
    - **`SQLiteConnection` (`conn`):** Objeto a nivel de clase que representa la conexión a la base de datos SQLite. Se inicializa con una cadena de conexión que especifica la ruta del archivo de la base de datos (`Data Source=preguntas.db;Version=3;`).
    - **`conn.Open()`:** Establece la conexión con la base de datos.
    - **`SQLiteCommand` (`cmd`):** Objeto que representa un comando SQL a ejecutar..
    - **`cmd.ExecuteNonQuery()`:** Ejecuta un comando SQL que no devuelve filas (como `CREATE TABLE` o `INSERT`).
    - **`cmd.ExecuteReader()`:** Ejecuta un comando SQL que devuelve filas (como `SELECT`) y retorna un `SQLiteDataReader`.
    - **`SQLiteDataReader` (`dr`):** Se utiliza para leer los resultados fila por fila de una consulta `SELECT`.
    - **`dr.Read()`:** Avanza el `DataReader` a la siguiente fila. Retorna `True` si hay más filas, `False` si ha llegado al final.
    - **`dr.Close()`:** Cierra el `DataReader`, liberando recursos.
- **Variables de Soporte (No controles visuales):**
    - **`preguntaActual` (String):** Almacena el texto de la pregunta obtenida de la base de datos.
    - **`respuestaActual` (String):** Almacena la respuesta correcta obtenida de la base de datos para la pregunta actual.
- **Explicación de la Lógica:**
    - Se importa `System.Data.SQLite` y se declara una conexión (`conn`) a un archivo de base de datos `preguntas.db`.
    - En el evento `Form1_Load`:
        - La conexión se abre (`conn.Open()`).
        - Se crea un comando SQL para asegurarse de que la tabla `preguntas` (con columnas `pregunta` y `respuesta`) exista, ejecutando `cmd.ExecuteNonQuery()`.
        - Se ejecuta una consulta `SELECT * FROM preguntas LIMIT 1` para obtener la primera pregunta y respuesta de la base de datos.
        - Si `dr.Read()` devuelve `True` (hay una pregunta), `preguntaActual` y `respuestaActual` se actualizan con los valores de las columnas, y `lblPregunta.Text` muestra la pregunta.
        - El `DataReader` se cierra (`dr.Close()`).
    - Cuando se hace clic en `btnVerificar`:

        - El texto de `txtRespuesta` se compara con `respuestaActual` (ignorando mayúsculas/minúsculas con `.ToLower()`).
        - Se muestra un `MessageBox` indicando si la respuesta fue "Correcto" o "Incorrecto".

Programa 138. Juego de preguntas con imágenes

- **Explicación del Ejercicio:** Este programa implementa un juego de preguntas donde la **pregunta se presenta visualmente mediante una imagen** en un `PictureBox`, y el usuario debe introducir la respuesta en un `TextBox`.
- **Controles Necesarios:**
    - **PictureBox:**
        - **Nombre (Name):** `picPregunta`. Control que muestra la imagen de la pregunta.
        - **Propiedad `Image`:** Se le asigna un objeto `Image` (cargado desde los recursos o archivos) para mostrar la imagen de la pregunta actual.
        - **Utilización:** Es el elemento visual principal que presenta la pregunta al usuario.
    - **TextBox:**
        - **Nombre (Name):** `txtRespuesta`. Campo de entrada para la respuesta del usuario.
        - **Propiedad `Text`:** Contiene la respuesta ingresada por el usuario.
        - **Utilización:** Permite al usuario introducir su respuesta.
    - **Button:**
        - **Nombre (Name):** `btnVerificar`.
        - **Texto (Text):** "Verificar".
        - **Evento `Click`:** `btnVerificar_Click`. Verifica la respuesta y avanza a la siguiente pregunta (cargando una nueva imagen).
        - **Utilización:** Dispara la verificación de la respuesta.
- **Recursos Adicionales:**
    - Se requiere tener archivos de imagen para las preguntas (ej., `pregunta1.png`, `pregunta2.png`) agregados a los **recursos del proyecto** (en Visual Studio: `My Project > Resources`). Esto permite acceder a ellas programáticamente mediante `My.Resources.ResourceManager.GetObject()`.
- **Variables de Soporte (No controles visuales):**
    - **`preguntas` (List(Of String)):** Lista de nombres de recursos de imagen que corresponden a las preguntas.
    - **`respuestas` (List(Of String)):** Lista de respuestas correctas correspondientes a las preguntas.
    - **`indice` (Integer):** Rastrea la pregunta actual.
- **Explicación de la Lógica:**
    - En `Form1_Load`, `picPregunta.Image` se carga con la primera imagen de pregunta (`preguntas(0)`), y `indice` se inicializa a 0.
    - Cuando se hace clic en `btnVerificar`:

- La respuesta del usuario (`txtRespuesta.Text`) se compara con `respuestas(indice)` (ignorando mayúsculas/minúsculas).
- Se muestra un `MessageBox` indicando si la respuesta es "Correcta" o "Incorrecta".
- `indice` se incrementa para avanzar a la siguiente pregunta.
- Si `indice` es menor que `preguntas.Count` (hay más preguntas), se carga la siguiente imagen (`My.Resources.ResourceManager.GetObject(preguntas(indice))`) en `picPregunta.Image`, y `txtRespuesta` se limpia.
- Si no hay más preguntas, se muestra un `MessageBox` de fin de juego.

Programa 139. Juego de preguntas con sonidos

- **Explicación del Ejercicio:** Este programa implementa un juego de preguntas donde la **pista para la respuesta es un sonido**, que el usuario puede reproducir. El usuario escucha el sonido y luego intenta adivinar lo que representa.
- **Controles Necesarios:**
  - **Button:**
    - **Nombre (Name):** `btnReproducir`.
    - **Texto (Text):** "Reproducir Sonido". Texto visible en el botón.
    - **Evento `Click`:** `btnReproducir_Click`. Se activa al hacer clic. Contiene la lógica para reproducir el archivo de sonido.
    - **Utilización:** Permite al usuario escuchar la pista de audio.
  - **TextBox:**
    - **Nombre (Name):** `txtRespuesta`. Campo de entrada para la respuesta del usuario.
    - **Propiedad `Text`:** Contiene la respuesta ingresada por el usuario.
    - **Utilización:** Permite al usuario introducir su respuesta basada en el sonido.
  - **Button:**
    - **Nombre (Name):** `btnVerificar`.
    - **Texto (Text):** "Verificar".
    - **Evento `Click`:** `btnVerificar_Click`. Se activa al hacer clic. Contiene la lógica para comparar la respuesta del usuario con la respuesta correcta.
    - **Utilización:** Dispara la verificación de la respuesta.
- **Recursos Adicionales:**
  - Se requiere tener un archivo de sonido (ej., `sonido1.wav`) en la misma carpeta que el ejecutable del programa.
- **Importación de Namespace:**
  - **`Imports System.Media`:** Necesario para utilizar la clase `SoundPlayer`.
- **Clases y Métodos de Audio:**
  - **`SoundPlayer` (`player`):** Una clase para reproducir archivos de sonido en formato WAV.
  - **`player.Play()`:** Inicia la reproducción del sonido cargado.
- **Variables de Soporte (No controles visuales):**
  - **`sonido` (String):** Almacena la ruta al archivo de sonido (ej., "sonido1.wav").

    - **`respuestaCorrecta` (String):** Almacena la respuesta esperada para el sonido.
- **Explicación de la Lógica:**
    - Se importa `System.Media` y se declaran las variables `sonido` (con la ruta del archivo WAV) y `respuestaCorrecta` a nivel de clase.
    - Cuando se hace clic en `btnReproducir`:
        - Se crea una nueva instancia de `SoundPlayer` con la ruta del archivo `sonido`.
        - `player.Play()` reproduce el sonido.
    - Cuando se hace clic en `btnVerificar`:
        - La respuesta ingresada en `txtRespuesta` se compara con `respuestaCorrecta` (ignorando mayúsculas/minúsculas con `.ToLower()`).
        - Se muestra un `MessageBox` indicando si la respuesta fue "Correcto" o "Incorrecto".

Programa 140. Juego de preguntas multijugador

- **Explicación del Ejercicio:** Este programa simula un **juego de preguntas para dos jugadores**, llevando la puntuación de cada uno de ellos. Ambos jugadores responden a la misma pregunta y se registran sus puntajes de forma independiente.
- **Controles Necesarios:**
    - **Label:**
        - **Nombre (Name):** `lblPregunta`. Identificador de la etiqueta que muestra la pregunta actual.
        - **Propiedad `Text`:** Se inicializa con la primera pregunta.
        - **Utilización:** Muestra la pregunta a ambos jugadores.
    - **TextBox:**
        - **Nombre (Name):** `txtRespuesta1`. Campo de entrada para la respuesta del Jugador 1.
        - **Propiedad `Text`:** Contiene la respuesta ingresada por el Jugador 1, que se compara con la respuesta correcta.
        - **Utilización:** Permite al Jugador 1 introducir su respuesta.
    - **TextBox:**
        - **Nombre (Name):** `txtRespuesta2`. Campo de entrada para la respuesta del Jugador 2.
        - **Propiedad `Text`:** Contiene la respuesta ingresada por el Jugador 2, que se compara con la respuesta correcta.
        - **Utilización:** Permite al Jugador 2 introducir su respuesta.
    - **Button:**
        - **Nombre (Name):** `btnVerificar`. Identificador del botón.
        - **Texto (Text):** "Verificar". Texto visible en el botón.
        - **Evento `Click`:** `btnVerificar_Click`. Se activa al hacer clic. Contiene la lógica para comparar las respuestas de ambos jugadores y actualizar sus puntajes.

        - **Utilización:** Dispara la verificación de las respuestas y la actualización de los puntajes.
    - **Label:**
        - **Nombre (Name):** `lblPuntaje`. Identificador de la etiqueta que muestra las puntuaciones de ambos jugadores.
        - **Propiedad `Text`:** Se actualiza para mostrar el puntaje de cada jugador (ej., "Jugador1: 1 - Jugador2: 0").
        - **Utilización:** Informa a ambos jugadores sobre su puntuación actual.
- **Variables de Soporte (No controles visuales):**
    - **`preguntas` (List(Of String)):** Lista a nivel de clase que almacena las preguntas del juego.
    - **`respuestas` (List(Of String)):** Lista a nivel de clase que almacena las respuestas correctas.
    - **`puntaje1` (Integer):** Variable a nivel de clase que almacena la puntuación del Jugador 1, inicializada a 0.
    - **`puntaje2` (Integer):** Variable a nivel de clase que almacena la puntuación del Jugador 2, inicializada a 0.
- **Explicación de la Lógica:**
    - En el evento `Form1_Load`, `lblPregunta` se establece con la primera pregunta de la lista `preguntas`, y `lblPuntaje` se inicializa mostrando "Jugador1: 0 - Jugador2: 0".
    - Cuando se hace clic en `btnVerificar`, el evento `btnVerificar_Click` se activa.
    - Se compara la respuesta de `txtRespuesta1.Text` (convertida a minúsculas) con la respuesta correcta de la pregunta actual (`respuestas(0).ToLower()`). Si coinciden, `puntaje1` se incrementa.
    - De manera similar, se compara la respuesta de `txtRespuesta2.Text` con la respuesta correcta. Si coinciden, `puntaje2` se incrementa.
    - `lblPuntaje.Text` se actualiza para mostrar las puntuaciones actuales de `puntaje1` y `puntaje2`.
    - **Nota:** El código proporcionado solo maneja una única pregunta y su respuesta (`preguntas(0)`, `respuestas(0)`). Para un juego multijugador completo, se necesitaría una lógica adicional para avanzar a la siguiente pregunta y limpiar los `TextBox` de respuesta, de forma similar a otros juegos de preguntas.

Programa 141. Juego de preguntas con ranking

- **Explicación del Ejercicio:** Este programa implementa un juego de preguntas que lleva un registro del puntaje actual del jugador y lo muestra en un ranking. El juego avanza a la siguiente pregunta después de verificar la respuesta y actualiza la lista de puntajes en tiempo real.
- **Controles Necesarios:**
    - **Label:**
        - **Nombre (Name):** `lblPregunta`.

        - **Propiedad `Text`:** Muestra la pregunta actual del juego. Se actualiza con la pregunta correspondiente de la lista `preguntas`.
        - **Utilización:** Presenta la pregunta al usuario.
    - **TextBox:**
        - **Nombre (Name):** `txtRespuesta`.
        - **Propiedad `Text`:** Contiene la respuesta ingresada por el usuario.
        - **Utilización:** Permite al usuario introducir su respuesta.
    - **Button:**
        - **Nombre (Name):** `btnVerificar`.
        - **Propiedad `Text`:** "Verificar".
        - **Evento `Click`:** `btnVerificar_Click`. Se activa cuando el usuario hace clic en el botón. Contiene la lógica para comprobar la respuesta, actualizar el puntaje actual (`puntajeActual`), añadirlo a la lista de `puntajes`, limpiar y repoblar el `lstRanking`, y avanzar a la siguiente pregunta.
        - **Utilización:** Dispara la verificación de la respuesta y la actualización del ranking.
    - **ListBox:**
        - **Nombre (Name):** `lstRanking`.
        - **Colección `Items`:** Almacena los puntajes de las rondas o el puntaje acumulado `puntajeActual`. Se limpia y se repuebla con los puntajes registrados hasta el momento.
        - **Método `Items.Clear()`:** Se llama para borrar todos los elementos existentes en la lista antes de volver a llenarla con los puntajes actualizados.
        - **Método `Items.Add()`:** Añade cada puntaje individual (`"Puntaje: " & p`) a la lista.
        - **Utilización:** Muestra el historial de puntajes del jugador.
- **Explicación de la Lógica:**
    - Se declaran a nivel de clase las listas `preguntas` y `respuestas` (`List(Of String)`) para el contenido del juego. `puntajes` (`List(Of Integer)`) almacena los puntajes de las rondas, e `indice` y `puntajeActual` (`Integer`) siguen el progreso y el puntaje actual del jugador.
    - En el evento `Form1_Load`, `lblPregunta.Text` se inicializa con la primera pregunta de la lista `preguntas` utilizando el `indice`.
    - Cuando se hace clic en `btnVerificar`:
        - La respuesta ingresada en `txtRespuesta.Text` se compara (ignorando mayúsculas/minúsculas con `.ToLower()`) con la respuesta correcta almacenada en `respuestas(indice)`. Si la respuesta es correcta, `puntajeActual` se incrementa.
        - El `puntajeActual` se añade a la lista `puntajes`.
        - `lstRanking` se limpia completamente (`lstRanking.Items.Clear()`) y luego se repuebla en un bucle `For Each` con todos los puntajes registrados hasta el momento en la lista `puntajes`.

- El `indice` se incrementa para pasar a la siguiente pregunta. Si todavía quedan preguntas (`indice < preguntas.Count`), `lblPregunta.Text` se actualiza con el texto de la siguiente pregunta. (Se sugiere manejar el fin del juego si no hay más preguntas).

Programa 142. Juego de preguntas con exportación de resultados

- **Explicación del Ejercicio:** Este programa permite exportar los resultados de un juego de preguntas, que se muestran en un `ListBox`, a un archivo de texto en el disco duro.
- **Controles Necesarios:**
  - **Button:**
    - **Nombre (Name):** `btnExportar`.
    - **Propiedad `Text`:** "Exportar Resultados".
    - **Evento `Click`:** `btnExportar_Click`. Se activa al hacer clic. Contiene la lógica para definir el nombre del archivo de texto, escribir todos los elementos del `ListBox` como líneas separadas en ese archivo y mostrar un mensaje de confirmación al usuario.
    - **Utilización:** Inicia el proceso de exportación de los resultados del juego a un archivo de texto.
  - **ListBox:**
    - **Nombre (Name):** `lstResultados`.
    - **Colección `Items`:** Contiene los resultados del juego (cada uno como una cadena de texto) que se exportarán al archivo.
    - **Utilización:** Muestra los resultados internos del juego que se van a exportar.
- **Importación de Namespace:**
  - **`Imports System.IO`:** Es necesario para utilizar las clases y métodos relacionados con la manipulación de archivos, como `File.WriteAllLines()`.
- **Explicación de la Lógica:**
  - Cuando se hace clic en `btnExportar`, el evento `btnExportar_Click` se ejecuta.
  - `Dim archivo As String = "resultados.txt"` define la ruta y el nombre del archivo donde se guardarán los resultados.
  - `File.WriteAllLines(archivo, lstResultados.Items.Cast(Of String)())` es la función clave. Este método estático de la clase `File` escribe todos los elementos de la colección `lstResultados.Items` como líneas separadas en el archivo especificado. La función `Cast(Of String)()` es necesaria para convertir la colección genérica `Items` a una colección de cadenas, ya que `WriteAllLines` espera un array o una enumeración de `String`.
  - Finalmente, `MessageBox.Show()` muestra un cuadro de diálogo confirmando que los resultados se han exportado y la ubicación del archivo.

Programa 143. Juego de preguntas con importación de resultados

- **Explicación del Ejercicio:** Este programa permite importar resultados de un juego de preguntas desde un archivo de texto existente y mostrarlos en un `ListBox` en el formulario.

- **Controles Necesarios:**
    - **Button:**
        - **Nombre (Name):** `btnImportar`.
        - **Propiedad `Text`:** "Importar Resultados".
        - **Evento `Click`:** `btnImportar_Click`. Se activa al hacer clic. Contiene la lógica para abrir un `OpenFileDialog`, y si el usuario selecciona un archivo, limpia el `ListBox` actual y carga todos los datos del archivo en él.
        - **Utilización:** Inicia el proceso de importación de resultados desde un archivo de texto.
    - **ListBox:**
        - **Nombre (Name):** `lstResultados`.
        - **Colección `Items`:** Se limpia antes de la importación para eliminar cualquier dato previo y luego se rellena con las líneas leídas del archivo de texto.
        - **Método `Items.Clear()`:** Vacía la lista de resultados antes de cargar los nuevos datos importados.
        - **Método `Items.AddRange()`:** Añade todos los elementos de un array de cadenas al `ListBox` de una sola vez, lo que es eficiente para cargar múltiples líneas.
        - **Utilización:** Muestra los resultados importados del juego al usuario.
    - **OpenFileDialog:**
        - **Nombre (Name):** `OpenFileDialog1`.
        - **Método `ShowDialog()`:** Abre el cuadro de diálogo estándar de Windows que permite al usuario navegar y seleccionar un archivo. Retorna `DialogResult.OK` si el usuario selecciona un archivo y hace clic en "Abrir".
        - **Propiedad `FileName`:** Contiene la ruta completa del archivo que el usuario ha seleccionado.
        - **Utilización:** Proporciona una interfaz gráfica para que el usuario busque y seleccione el archivo de texto con los resultados a importar.
- **Importación de Namespace:**
    - **`Imports System.IO`:** Es necesario para utilizar los métodos de lectura de archivos, como `File.ReadAllLines()`.
- **Explicación de la Lógica:**
    - Cuando se hace clic en `btnImportar`, `OpenFileDialog1.ShowDialog()` se abre para que el usuario elija un archivo.
    - Si el usuario selecciona un archivo y presiona "Abrir" (`DialogResult.OK`), la condición `If OpenFileDialog1.ShowDialog() = DialogResult.OK Then` se cumple.
    - Dentro de esta condición:
        - `lstResultados.Items.Clear()` limpia todos los elementos actuales del `ListBox`.
        - `File.ReadAllLines(OpenFileDialog1.FileName)` lee todas las líneas de texto del archivo seleccionado y las devuelve como un array de cadenas.

        - `lstResultados.Items.AddRange(File.ReadAllLines(OpenFileDialog1.FileName))` añade todas estas líneas (el array de cadenas) como elementos al `ListBox`, actualizando la lista de resultados visible.

Programa 144. Juego de preguntas con estadísticas

- **Explicación del Ejercicio:** Este programa muestra estadísticas en tiempo real de un juego de preguntas, como el número de respuestas correctas y el total de intentos, en una etiqueta durante el desarrollo del juego.
- **Controles Necesarios:**
    - **Label:**
        - **Nombre (Name):** `lblPregunta`.
        - **Propiedad `Text`:** Muestra la pregunta actual del juego. Se actualiza con el texto de la pregunta correspondiente a `indice`, o con "Juego terminado" al finalizar.
        - **Utilización:** Presenta la pregunta al usuario.
    - **TextBox:**
        - **Nombre (Name):** `txtRespuesta`.
        - **Propiedad `Text`:** Contiene la respuesta ingresada por el usuario.
        - **Utilización:** Permite al usuario introducir su respuesta a la pregunta.
    - **Button:**
        - **Nombre (Name):** `btnVerificar`.
        - **Propiedad `Text`:** "Verificar".
        - **Evento `Click`:** `btnVerificar_Click`. Se activa al hacer clic. Contiene la lógica para incrementar el contador de intentos, verificar si la respuesta es correcta (incrementando el contador de correctas si es así), avanzar a la siguiente pregunta y actualizar las estadísticas en `lblEstadisticas`.
        - **Utilización:** Dispara la verificación de la respuesta y la actualización de las estadísticas del juego.
    - **Label:**
        - **Nombre (Name):** `lblEstadisticas`.
        - **Propiedad `Text`:** Muestra el número de respuestas correctas y el total de intentos en un formato legible (ej., "Correctas: 1 / Intentos: 2").
        - **Utilización:** Informa al usuario sobre su rendimiento y el progreso del juego.
- **Explicación de la Lógica:**
    - Se declaran a nivel de clase las listas `preguntas` y `respuestas` (`List(Of String)`) que contienen el contenido del juego. Las variables `correctas`, `intentos` e `indice` (`Integer`) se utilizan para llevar el seguimiento de las respuestas correctas, el total de intentos y la pregunta actual, respectivamente.
    - En el evento `Form1_Load`, `lblPregunta.Text` se inicializa con la primera pregunta de la lista `preguntas` utilizando el `indice`.
    - Cuando se hace clic en `btnVerificar`:
        - La variable `intentos` se incrementa en 1.
        - La respuesta del usuario en `txtRespuesta.Text` se compara (después de convertirla a minúsculas con `.ToLower()`) con la respuesta correcta

almacenada en `respuestas(indice)`. Si coinciden, la variable `correctas` se incrementa en 1.
- El `indice` se incrementa para avanzar a la siguiente pregunta.
- Se verifica si aún quedan preguntas por responder (`If indice < preguntas.Count Then`). Si es así, `lblPregunta.Text` se actualiza con la siguiente pregunta y `txtRespuesta` debería limpiarse (práctica recomendada). Si no quedan preguntas, `lblPregunta.Text` muestra "Juego terminado".
- Finalmente, `lblEstadisticas.Text` se actualiza para mostrar el conteo actual de `correctas` y `intentos`.

Programa 145. Juego de preguntas con gráficos

- **Explicación del Ejercicio:** Este programa visualiza los resultados de un juego de preguntas (respuestas correctas e incorrectas) utilizando el control `Chart` para mostrar un gráfico de columnas.
- **Controles Necesarios:**
  - **Chart:**
    - **Nombre (Name):** `chartResultados`.
    - **Colección `Series`:** Contiene las series de datos que se dibujarán en el gráfico.
    - **Método `Series.Clear()`:** Borra todas las series de datos existentes en el gráfico antes de añadir nuevas, asegurando que el gráfico se actualice correctamente.
    - **Método `Series.Add()`:** Añade una nueva serie de datos al gráfico. La serie se puede nombrar ("Resultados" en este caso) y su tipo de gráfico (`ChartType`) se puede establecer (ej., `SeriesChartType.Column`).
    - **Método `Points.AddXY()`:** Añade un punto de datos a una serie. Permite especificar la etiqueta para el eje X (ej., "Correctas") y el valor para el eje Y (ej., `correctas`).
    - **Utilización:** Muestra una representación gráfica de los resultados del juego (número de respuestas correctas e incorrectas).
  - **Button:**
    - **Nombre (Name):** `btnActualizar`.
    - **Propiedad `Text`:** "Actualizar Gráfico".
    - **Evento `Click`:** `btnActualizar_Click`. Se activa al hacer clic. Limpia el gráfico, crea una nueva serie de datos, le añade los puntos de datos de "Correctas" e "Incorrectas" y la muestra.
    - **Utilización:** Dispara la actualización y visualización del gráfico.
- **Requisito Adicional:**
  - Asegurarse de que el control `Chart` esté disponible en el `Toolbox` de Visual Studio. Puede ser necesario añadir la referencia `System.Windows.Forms.DataVisualization` al proyecto.
- **Importación de Namespace:**

    - **`Imports System.Windows.Forms.DataVisualization.Charting`:** Es necesario para utilizar las clases relacionadas con el control `Chart`.
- **Explicación de la Lógica:**
    - Se declaran a nivel de clase las variables `correctas` e `incorrectas` (`Integer`), que en este ejemplo tienen valores simulados (3 y 2, respectivamente) para los datos a graficar.
    - Cuando se hace clic en `btnActualizar`:
        - `chartResultados.Series.Clear()` borra cualquier serie de datos existente en el gráfico, preparando el lienzo para el nuevo dibujo.
        - `chartResultados.Series.Add("Resultados")` añade una nueva serie de datos con el nombre "Resultados" al gráfico.
        - `chartResultados.Series("Resultados").Points.AddXY("Correctas", correctas)` añade un punto de datos a la serie. La etiqueta en el eje X será "Correctas" y su valor en el eje Y será el contenido de la variable `correctas`.
        - `chartResultados.Series("Resultados").Points.AddXY("Incorrectas", incorrectas)` añade otro punto de datos para "Incorrectas" y el valor de `incorrectas`. Esto visualiza ambos valores como columnas en el gráfico.

Programa 146. Juego de preguntas con personalización

- **Explicación del Ejercicio:** Este programa permite al usuario añadir sus propias preguntas a una lista (`ListBox`), lo que le permite personalizar el contenido del juego de preguntas.
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtPregunta`.
        - **Propiedad `Text`:** Contiene el texto de la pregunta que el usuario desea agregar a la lista.
        - **Método `Clear()`:** Se utiliza (aunque no está explícitamente en la fuente, es una buena práctica) para vaciar el `TextBox` después de agregar la pregunta, dejándolo listo para una nueva entrada.
        - **Utilización:** Permite al usuario introducir el texto de una nueva pregunta.
    - **Button:**
        - **Nombre (Name):** `btnAgregarPregunta`.
        - **Propiedad `Text`:** "Agregar Pregunta".
        - **Evento `Click`:** `btnAgregarPregunta_Click`. Se activa cuando el usuario pulsa el botón. Contiene la lógica para añadir el texto de `txtPregunta` como un nuevo elemento al `lstPreguntas`.
        - **Utilización:** Dispara la acción de añadir una nueva pregunta a la lista.
    - **ListBox:**
        - **Nombre (Name):** `lstPreguntas`.
        - **Colección `Items`:** Almacena y visualiza las preguntas que el usuario ha añadido.

        - **Método `Items.Add()`:** Añade el texto de la pregunta como un nuevo elemento a la lista.
        - **Utilización:** Muestra las preguntas personalizadas que el usuario ha creado.
- **Explicación de la Lógica:**
    - Cuando se hace clic en `btnAgregarPregunta`:
        - `lstPreguntas.Items.Add(txtPregunta.Text)` toma el texto que el usuario ha introducido en el `TextBox txtPregunta` y lo añade como un nuevo elemento a la colección `Items` del `ListBox lstPreguntas`. Esto permite al usuario ver inmediatamente la pregunta que ha añadido.
        - Para que un juego de preguntas sea completo, también se necesitaría un `TextBox` adicional para la respuesta de esa pregunta y un mecanismo para almacenar tanto la pregunta como su respuesta de forma conjunta.

Programa 147. Juego de preguntas con guardado automático

- **Explicación del Ejercicio:** Este programa permite guardar automáticamente las respuestas del usuario en un archivo de texto cada cierto tiempo (`Timer`), además de ofrecer una opción para un guardado manual (`Button`).
- **Controles Necesarios:**
    - **TextBox:**
        - **Nombre (Name):** `txtRespuesta`.
        - **Propiedad `Text`:** Contiene la respuesta actual del usuario que se va a guardar. Es la fuente de datos para el guardado automático y manual.
        - **Utilización:** Permite al usuario introducir su respuesta a una pregunta y sirve como el origen del texto que se guardará.
    - **Button:**
        - **Nombre (Name):** `btnGuardar`.
        - **Propiedad `Text`:** "Guardar".
        - **Evento `Click`:** `btnGuardar_Click`. Se activa al hacer clic. Guarda el contenido actual de `txtRespuesta` en el archivo de texto y, opcionalmente, muestra un mensaje de confirmación al usuario.
        - **Utilización:** Permite al usuario realizar un guardado manual de su respuesta en cualquier momento.
    - **Timer:**
        - **Nombre (Name):** `Timer1`.
        - **Propiedad `Interval`:** 5000 (milisegundos). Se configura para disparar su evento `Tick` cada 5000 milisegundos (5 segundos).
        - **Evento `Tick`:** `Timer1_Tick`. Se activa periódicamente cada 5 segundos y contiene la lógica para guardar automáticamente el contenido de `txtRespuesta` en el archivo.
        - **Utilización:** Es el mecanismo para el guardado automático de la respuesta del usuario, asegurando que el progreso no se pierda.
- **Importación de Namespace:**
    - **`Imports System.IO`:** Es necesario para utilizar las clases y métodos relacionados con la manipulación de archivos, como `File.AppendAllText()`.

- **Explicación de la Lógica:**
    - El `Timer1` debe estar iniciado para que el guardado automático funcione (opcionalmente, en el evento `Form_Load`).
    - En cada disparo del evento `Timer1_Tick` (cada 5 segundos):
        - `File.AppendAllText("respuestas.txt", txtRespuesta.Text & Environment.NewLine)` añade el contenido actual de `txtRespuesta` (seguido de un salto de línea) al final del archivo "respuestas.txt". Si el archivo no existe, lo crea.
    - Cuando se hace clic en `btnGuardar`:
        - Se realiza la misma acción de `File.AppendAllText()` para guardar el texto manualmente.
        - Se puede añadir opcionalmente un `MessageBox.Show()` para confirmar el guardado manual al usuario.

Programa 148. Juego de preguntas con recuperación de partida

- **Explicación del Ejercicio:** Este programa permite guardar el progreso de un juego de preguntas (específicamente la posición de la pregunta actual) y recuperarlo posteriormente para continuar el juego desde ese punto.
- **Controles Necesarios:**
    - **Button:**
        - **Nombre (Name):** `btnGuardar`.
        - **Propiedad `Text`:** "Guardar Partida".
        - **Evento `Click`:** `btnGuardar_Click`. Se activa al hacer clic. Contiene la lógica para escribir el `indice` de la pregunta actual en un archivo de texto.
        - **Utilización:** Permite al usuario guardar su progreso en el juego en un archivo.
    - **Button:**
        - **Nombre (Name):** `btnCargar`.
        - **Propiedad `Text`:** "Cargar Partida".
        - **Evento `Click`:** `btnCargar_Click`. Se activa al hacer clic. Contiene la lógica para verificar si existe un archivo de partida guardada, leer el `indice` desde ese archivo y actualizar la `lblPregunta` para reanudar el juego.
        - **Utilización:** Permite al usuario reanudar una partida guardada previamente.
    - **Label:**
        - **Nombre (Name):** `lblPregunta`.
        - **Propiedad `Text`:** Muestra la pregunta actual del juego. Se actualiza con la pregunta correspondiente al `indice` cargado.
        - **Utilización:** Presenta la pregunta al usuario.
- **Importación de Namespace:**
    - **`Imports System.IO`:** Es necesario para utilizar las clases y métodos relacionados con la manipulación de archivos, como `File.WriteAllText()` y `File.ReadAllText()`.
- **Explicación de la Lógica:**

- Se declaran a nivel de clase la lista `preguntas` (`List(Of String)`) que contiene el texto de las preguntas, y la variable `indice` (`Integer`) que rastrea la posición de la pregunta actual en la lista.
- Cuando se hace clic en `btnGuardar`:
  - `File.WriteAllText("partida.txt", indice.ToString())` escribe el valor actual de la variable `indice` (convertido a cadena) en un archivo de texto llamado "partida.txt".
  - (Opcionalmente, `MessageBox.Show("Partida guardada.")` confirmaría el guardado al usuario).
- Cuando se hace clic en `btnCargar`:
  - `File.Exists("partida.txt")` verifica si el archivo "partida.txt" existe en el sistema.
  - Si el archivo existe:
    - `indice = CInt(File.ReadAllText("partida.txt"))` lee el contenido del archivo (que es el `indice` guardado), lo convierte a un entero (`CInt()`) y lo asigna de nuevo a la variable `indice`.
    - `lblPregunta.Text = preguntas(indice)` actualiza la `lblPregunta` con la pregunta correspondiente a la posición recién cargada.
    - (Opcionalmente, `MessageBox.Show("Partida cargada.")` confirmaría la carga).
  - Si el archivo no existe, se muestra un `MessageBox` indicando que no hay partida guardada (práctica recomendada).

Programa 149. Juego de preguntas con dificultad progresiva

- **Explicación del Ejercicio:** Este programa implementa un juego de preguntas que aumenta su dificultad a medida que el jugador avanza de nivel. El cambio de dificultad se simula cargando diferentes conjuntos de preguntas (listas) según el nivel actual.
- **Controles Necesarios:**
  - **Label:**
    - **Nombre (Name):** `lblPregunta`.
    - **Propiedad `Text`:** Muestra la pregunta actual. Se actualiza con la primera pregunta del nivel de dificultad correspondiente a medida que el jugador avanza.
    - **Utilización:** Presenta la pregunta al usuario.
  - **TextBox:**
    - **Nombre (Name):** `txtRespuesta`.
    - **Propiedad `Text`:** Contiene la respuesta ingresada por el usuario.
    - **Método `Clear()`:** Se utiliza (aunque no está explícitamente en la fuente, es una buena práctica) para limpiar el `TextBox` después de cada avance de nivel, preparándolo para la nueva pregunta.
    - **Utilización:** Permite al usuario introducir su respuesta a la pregunta actual.
  - **Button:**

        - **Nombre (Name):** `btnVerificar`.
        - **Propiedad `Text`:** "Verificar".
        - **Evento `Click`:** `btnVerificar_Click`. Se activa al hacer clic. Contiene la lógica para avanzar el nivel del juego (`nivel`) y cargar la primera pregunta del siguiente nivel de dificultad.
        - **Utilización:** Desencadena el avance de nivel en el juego.
- **Explicación de la Lógica:**
    - Se definen a nivel de clase listas de preguntas separadas para diferentes niveles de dificultad: `preguntasFacil`, `preguntasMedio`, y `preguntasDificil` (`List(Of String)`). La variable `nivel` (`Integer`) se inicializa a 1 (para "Fácil") y controla la dificultad actual del juego.
    - En el evento `Form1_Load`, `lblPregunta.Text` se inicializa con la primera pregunta del nivel "Fácil" (`preguntasFacil(0)`).
    - Cuando se hace clic en `btnVerificar`:
        - (Aunque la fuente no lo muestra, en una implementación real se debería **verificar la respuesta** del usuario para la pregunta actual antes de avanzar de nivel).
        - Un bloque `Select Case nivel` evalúa el nivel actual del juego.
        - **Caso 1 (Nivel Fácil):** Si `nivel` es 1, `lblPregunta.Text` se actualiza con la primera pregunta del nivel "Medio" (`preguntasMedio(0)`), y la variable `nivel` se incrementa a 2.
        - **Caso 2 (Nivel Medio):** Si `nivel` es 2, `lblPregunta.Text` se actualiza con la primera pregunta del nivel "Difícil" (`preguntasDificil(0)`), y la variable `nivel` se incrementa a 3.
        - **Caso 3 (Nivel Difícil):** Si `nivel` es 3, `lblPregunta.Text` se establece en "Juego terminado", indicando que el jugador ha completado todos los niveles.
        - `txtRespuesta.Clear()` se llama para limpiar el `TextBox` después de cada avance de nivel, preparando la interfaz para la siguiente pregunta.

Programa 150. Juego de preguntas con logros desbloqueables

- **Explicación del Ejercicio:** Este programa implementa un juego de preguntas que recompensa al jugador con "logros" (mensajes de texto) al alcanzar ciertos hitos de puntuación. Estos logros se muestran en un `ListBox` dedicado.
- **Controles Necesarios:**
    - **Label:**
        - **Nombre (Name):** `lblPregunta`.
        - **Propiedad `Text`:** Muestra la pregunta actual del juego.
        - **Utilización:** Presenta la pregunta al usuario.
    - **TextBox:**
        - **Nombre (Name):** `txtRespuesta`.
        - **Propiedad `Text`:** Contiene la respuesta ingresada por el usuario.
        - **Utilización:** Permite al usuario introducir su respuesta a la pregunta actual.
    - **Button:**

        - **Nombre (Name):** `btnVerificar`.
        - **Propiedad `Text`:** "Verificar".
        - **Evento `Click`:** `btnVerificar_Click`. Se activa al hacer clic. Contiene la lógica para verificar la respuesta del usuario, incrementar el puntaje, y si el `puntaje` alcanza ciertos umbrales, añadir los logros correspondientes al `lstLogros`.
        - **Utilización:** Desencadena la verificación de la respuesta y la comprobación de logros.
    - **ListBox:**
        - **Nombre (Name):** `lstLogros`.
        - **Colección `Items`:** Almacena y visualiza los mensajes de los logros que el jugador ha desbloqueado.
        - **Método `Items.Add()`:** Añade el texto de un logro a la lista.
        - **Utilización:** Muestra al jugador los logros que ha conseguido durante el juego.
- **Explicación de la Lógica:**
    - Se declaran a nivel de clase las listas `preguntas` y `respuestas` (`List(Of String)`) que contienen el contenido del juego, y la variable `puntaje` (`Integer`) que almacena la puntuación del jugador.
    - En el evento `Form1_Load`, `lblPregunta.Text` se inicializa con la primera pregunta de la lista `preguntas` (`preguntas(0)`).
    - Cuando se hace clic en `btnVerificar`:
        - (Nota: La fuente proporcionada utiliza `lstLogros.Items.Count` como índice para acceder a las respuestas, lo cual es una implementación inusual. Normalmente, se tendría una variable de clase (`indicePregunta` por ejemplo) que rastree la pregunta actual independientemente de los logros). Asumiendo que se tiene un `indice` correcto para la pregunta: la respuesta en `txtRespuesta.Text` se compara (ignorando mayúsculas/minúsculas) con la respuesta correcta.
        - Si la respuesta es correcta, la variable `puntaje` se incrementa.
        - Después de actualizar el puntaje, se verifican las condiciones para desbloquear logros:
            - Si `puntaje` es igual a 1, el logro "Primer acierto!" se añade a `lstLogros`.
            - Si `puntaje` es igual a 2, el logro "Dos aciertos seguidos!" se añade a `lstLogros`.
        - (Opcionalmente, se avanzaría a la siguiente pregunta y `txtRespuesta` se limpiaría para un juego completo).

Programa 151. Editor de imágenes simple

- **Explicación del Ejercicio:** Este programa permite al usuario cargar una imagen desde su equipo y aplicarle un filtro básico de escala de grises. La imagen modificada se muestra en el mismo `PictureBox`.
- **Controles Necesarios:**

    - **PictureBox:**
        - **Nombre (Name):** `picImagen`.
        - **Propiedad `SizeMode`:** `StretchImage`. Esta propiedad es importante, ya que hace que la imagen se ajuste automáticamente para llenar el tamaño del `PictureBox`, evitando que se corte o se muestre en su tamaño original si es muy grande.
        - **Propiedad `Image`:** Almacena el objeto `Image` (la imagen original cargada o la imagen modificada con el filtro) que se muestra en el control.
        - **Utilización:** Muestra la imagen que el usuario carga y, posteriormente, la imagen con el filtro de escala de grises aplicado.
    - **Button:**
        - **Nombre (Name):** `btnCargar`.
        - **Propiedad `Text`:** "Cargar Imagen".
        - **Evento `Click`:** `btnCargar_Click`. Se activa cuando el usuario hace clic en el botón. Contiene la lógica para abrir un `OpenFileDialog` y cargar la imagen seleccionada en `picImagen`.
        - **Utilización:** Permite al usuario seleccionar un archivo de imagen desde su sistema para ser editado.
    - **Button:**
        - **Nombre (Name):** `btnGrises`.
        - **Propiedad `Text`:** "Convertir a Gris".
        - **Evento `Click`:** `btnGrises_Click`. Se activa al hacer clic. Contiene la lógica para aplicar el filtro de escala de grises a la imagen actual en `picImagen`.
        - **Utilización:** Aplica el filtro de escala de grises a la imagen que se muestra en el `PictureBox`.
    - **OpenFileDialog:**
        - **Nombre (Name):** `OpenFileDialog1`.
        - **Método `ShowDialog()`:** Abre el cuadro de diálogo estándar de Windows que permite al usuario navegar y seleccionar un archivo. Retorna `DialogResult.OK` si el usuario selecciona un archivo y hace clic en "Abrir".
        - **Propiedad `FileName`:** Contiene la ruta completa del archivo que el usuario ha seleccionado.
        - **Utilización:** Proporciona una interfaz gráfica para que el usuario busque y seleccione el archivo de imagen a cargar.
- **Explicación de la Lógica:**
    - Cuando se hace clic en `btnCargar`:
        - Si `OpenFileDialog1.ShowDialog()` devuelve `DialogResult.OK`, la línea `picImagen.Image = Image.FromFile(OpenFileDialog1.FileName)` carga el archivo de imagen seleccionado por el usuario en el `PictureBox`.
    - Cuando se hace clic en `btnGrises`:

        - Primero, se verifica que `picImagen.Image` no sea `Nothing` para asegurar que hay una imagen cargada.
        - `Dim bmp As New Bitmap(picImagen.Image)` crea una nueva instancia de `Bitmap` a partir de la imagen actual en `picImagen`. Esto es necesario porque el objeto `Image` directamente no siempre es editable; `Bitmap` proporciona métodos para la manipulación de píxeles.
        - Dos bucles anidados (`For x As Integer = 0 To bmp.Width - 1` y `For y As Integer = 0 To bmp.Height - 1`) iteran sobre cada píxel de la imagen.
        - Dentro de los bucles, `Dim c As Color = bmp.GetPixel(x, y)` obtiene el objeto `Color` del píxel actual.
        - `Dim gris As Integer = CInt((c.R + c.G + c.B) / 3)` calcula el valor de gris promediando los componentes Rojo, Verde y Azul del color original del píxel. `CInt()` convierte el resultado a un entero.
        - `bmp.SetPixel(x, y, Color.FromArgb(gris, gris, gris))` establece el color del píxel actual a una nueva tonalidad de gris. `Color.FromArgb(gris, gris, gris)` crea un color con los mismos valores para Rojo, Verde y Azul, resultando en un gris.
        - Finalmente, `picImagen.Image = bmp` asigna la imagen `Bitmap` modificada (ahora en escala de grises) de nuevo al `PictureBox` para su visualización.

Programa 152. Editor de audio simple

- **Explicación del Ejercicio:** Este programa permite al usuario cargar un archivo de audio (WAV) desde su sistema y luego reproducirlo con un botón.
- **Controles Necesarios:**
    - **Button:**
        - **Nombre (Name):** `btnCargar`.
        - **Propiedad `Text`:** "Cargar Audio".
        - **Evento `Click`:** `btnCargar_Click`. Se activa cuando el usuario pulsa el botón. Contiene la lógica para abrir un `OpenFileDialog`, y si se selecciona un archivo, inicializa el objeto `SoundPlayer` con la ruta del archivo de audio elegido.
        - **Utilización:** Permite al usuario seleccionar un archivo de audio (formato WAV) desde su disco duro.
    - **Button:**
        - **Nombre (Name):** `btnReproducir`.
        - **Propiedad `Text`:** "Reproducir".
        - **Evento `Click`:** `btnReproducir_Click`. Se activa cuando el usuario pulsa el botón. Contiene la lógica para, si ya se ha cargado un archivo de audio, reproducir el sonido.
        - **Utilización:** Reproduce el archivo de audio que ha sido cargado previamente.
    - **OpenFileDialog:**
        - **Nombre (Name):** `OpenFileDialog1`.

        - **Método `ShowDialog()`:** Abre el cuadro de diálogo estándar de Windows que permite al usuario navegar y seleccionar un archivo. Retorna `DialogResult.OK` si el usuario selecciona un archivo y hace clic en "Abrir".
        - **Propiedad `FileName`:** Contiene la ruta completa del archivo de audio que el usuario ha seleccionado.
        - **Utilización:** Proporciona una interfaz gráfica para que el usuario busque y seleccione un archivo de audio (WAV) a cargar.
- **Importación de Namespace:**
    - **`Imports System.Media`:** Es necesario para utilizar la clase `SoundPlayer`, que es la encargada de reproducir los archivos de audio en formato WAV.
- **Explicación de la Lógica:**
    - Se declara una instancia de la clase `SoundPlayer` llamada `player` a nivel de clase (`Dim player As SoundPlayer`) para que pueda ser accesible por ambos eventos de botón.
    - En el evento `btnCargar_Click`:
        - Cuando el `OpenFileDialog1.ShowDialog()` se abre y el usuario selecciona un archivo y presiona "Abrir" (`DialogResult.OK`), la línea `player = New SoundPlayer(OpenFileDialog1.FileName)` inicializa el objeto `player` con la ruta completa del archivo de audio elegido.
    - En el evento `btnReproducir_Click`:
        - Primero, se verifica que el objeto `player` no sea `Nothing` (es decir, que ya se haya cargado un archivo de audio).
        - Si `player` no es nulo, `player.Play()` inicia la reproducción del sonido.

Programa 153. Reproductor de música básico

- **Explicación del Ejercicio:** Este programa implementa un reproductor de música con funciones básicas de cargar, reproducir y detener canciones. Para lograr esto, utiliza el control `AxWindowsMediaPlayer`, que es una envoltura del reproductor de Windows Media Player.
- **Controles Necesarios:**
    - **Button:**
        - **Nombre (Name):** `btnCargar`.
        - **Propiedad `Text`:** "Cargar Música".
        - **Evento `Click`:** `btnCargar_Click`. Se activa cuando el usuario pulsa el botón. Contiene la lógica para abrir un `OpenFileDialog` y, si se selecciona un archivo, establecer la propiedad `URL` del `AxWindowsMediaPlayer` con la ruta de la canción elegida.
        - **Utilización:** Permite al usuario seleccionar un archivo de música (MP3, WAV, etc.) desde su sistema.
    - **Button:**
        - **Nombre (Name):** `btnPlay`.
        - **Propiedad `Text`:** "Play".

        - **Evento `Click`:** `btnPlay_Click`. Se activa cuando el usuario pulsa el botón. Contiene la lógica para iniciar la reproducción del archivo multimedia cargado en el `AxWindowsMediaPlayer`.
        - **Utilización:** Inicia la reproducción de la canción actualmente cargada.
    - **Button:**
        - **Nombre (Name):** `btnStop`.
        - **Propiedad `Text`:** "Stop".
        - **Evento `Click`:** `btnStop_Click`. Se activa cuando el usuario pulsa el botón. Contiene la lógica para detener la reproducción del `AxWindowsMediaPlayer`.
        - **Utilización:** Detiene la reproducción de la canción actual.
    - **OpenFileDialog:**
        - **Nombre (Name):** `OpenFileDialog1`.
        - **Método `ShowDialog()`:** Abre el cuadro de diálogo estándar de Windows que permite al usuario navegar y seleccionar un archivo. Retorna `DialogResult.OK` si el usuario selecciona un archivo y hace clic en "Abrir".
        - **Propiedad `FileName`:** Contiene la ruta completa del archivo de música que el usuario ha seleccionado.
        - **Utilización:** Proporciona una interfaz gráfica para que el usuario busque y seleccione un archivo de música a cargar.
    - **AxWindowsMediaPlayer:**
        - **Nombre (Name):** `wmp` (se asume, aunque no está explícitamente nombrado en el fragmento de código, es el nombre común para este control).
        - **Propiedad `URL`:** Esta propiedad es clave. Se establece con la ruta completa del archivo multimedia (audio o video) que se desea cargar y reproducir.
        - **Propiedad `Ctlcontrols`:** Permite acceder a una interfaz que contiene métodos para controlar la reproducción multimedia, como `play()` y `stop()`.
        - **Utilización:** Es el control principal que incrusta el reproductor de Windows Media Player en el formulario y maneja la reproducción del audio.
- **Requisito Adicional:**
    - Este programa requiere que se añada el componente `AxWindowsMediaPlayer` al formulario. Esto se suele hacer arrastrándolo desde el "Cuadro de herramientas" de Visual Studio después de haber añadido una referencia al control COM apropiado (generalmente "Windows Media Player") en el proyecto.
- **Explicación de la Lógica:**
    - En el evento `btnCargar_Click`:
        - Cuando `OpenFileDialog1.ShowDialog()` se abre y el usuario selecciona un archivo y presiona "Abrir" (`DialogResult.OK`), la propiedad `wmp.URL` se establece con la `OpenFileDialog1.FileName` (la ruta del archivo de música elegido). Esto carga la música en el reproductor.
    - En el evento `btnPlay_Click`:

        - `wmp.Ctlcontrols.play()` invoca el método `play()` del control `AxWindowsMediaPlayer`, lo que inicia la reproducción de la canción cargada.
    - En el evento `btnStop_Click`:
        - `wmp.Ctlcontrols.stop()` invoca el método `stop()` del control `AxWindowsMediaPlayer`, lo que detiene la reproducción de la canción.

Programa 154. Reproductor de video básico

- **Explicación del Ejercicio:** Este programa permite al usuario cargar un archivo de video desde su sistema y reproducirlo automáticamente dentro del formulario. Utiliza el control `AxWindowsMediaPlayer` para manejar la reproducción.
- **Controles Necesarios:**
    - **Button:**
        - **Nombre (Name):** `btnCargar`.
        - **Propiedad `Text`:** "Cargar Video".
        - **Evento `Click`:** `btnCargar_Click`. Se activa cuando el usuario pulsa el botón. Contiene la lógica para abrir un `OpenFileDialog`, y si se selecciona un archivo, establecer la `URL` del `AxWindowsMediaPlayer` con la ruta del video elegido y comenzar la reproducción.
        - **Utilización:** Permite al usuario seleccionar un archivo de video (MP4, AVI, WMV, etc.) desde su sistema para cargarlo en el reproductor.
    - **AxWindowsMediaPlayer:**
        - **Nombre (Name):** `wmp` (se asume, ya que no está explícitamente nombrado en la fuente).
        - **Propiedad `URL`:** Esta propiedad es fundamental. Se establece con la ruta completa del archivo de video que se desea cargar y reproducir.
        - **Propiedad `Ctlcontrols`:** Permite acceder a una interfaz que contiene métodos para controlar la reproducción multimedia, como `play()`.
        - **Utilización:** Es el control principal que incrusta el reproductor de Windows Media Player en el formulario y se encarga de mostrar y reproducir el video.
    - **OpenFileDialog:**
        - **Nombre (Name):** `OpenFileDialog1`.
        - **Método `ShowDialog()`:** Abre el cuadro de diálogo estándar de Windows que permite al usuario navegar y seleccionar un archivo. Retorna `DialogResult.OK` si el usuario selecciona un archivo y hace clic en "Abrir".
        - **Propiedad `FileName`:** Contiene la ruta completa del archivo de video que el usuario ha seleccionado.
        - **Utilización:** Proporciona una interfaz gráfica para que el usuario busque y seleccione un archivo de video a cargar.
- **Explicación de la Lógica:**
    - En el evento `btnCargar_Click`:
        - Cuando `OpenFileDialog1.ShowDialog()` se abre y el usuario selecciona un archivo y presiona "Abrir" (`DialogResult.OK`):

- `wmp.URL = OpenFileDialog1.FileName` establece la propiedad `URL` del control `AxWindowsMediaPlayer` con la ruta completa del archivo de video elegido. Esto carga el video en el reproductor.
- `wmp.Ctlcontrols.play()` invoca el método `play()` inmediatamente, lo que hace que el video comience a reproducirse tan pronto como se carga.

Programa 155. Reproductor multimedia integrado

- **Explicación del Ejercicio:** Este programa combina las funcionalidades de los reproductores de audio y video en una sola interfaz. Permite al usuario cargar y reproducir archivos de ambos tipos (audio y video) utilizando filtros específicos en el `OpenFileDialog` para facilitar la selección.
- **Controles Necesarios:**
  - **Button:**
    - **Nombre (Name):** `btnCargarAudio`.
    - **Propiedad `Text`:** "Cargar Audio".
    - **Evento `Click`:** `btnCargarAudio_Click`. Se activa al hacer clic. Configura el filtro del `OpenFileDialog` para mostrar únicamente archivos de audio (WAV, MP3) y, si el usuario selecciona uno, carga su ruta en la propiedad `URL` del `AxWindowsMediaPlayer`.
    - **Utilización:** Permite al usuario seleccionar y cargar un archivo de audio para su reproducción.
  - **Button:**
    - **Nombre (Name):** `btnCargarVideo`.
    - **Propiedad `Text`:** "Cargar Video".
    - **Evento `Click`:** `btnCargarVideo_Click`. Se activa al hacer clic. Configura el filtro del `OpenFileDialog` para mostrar solo archivos de video (MP4, AVI) y, si el usuario selecciona uno, carga su ruta en la propiedad `URL` del `AxWindowsMediaPlayer`.
    - **Utilización:** Permite al usuario seleccionar y cargar un archivo de video para su reproducción.
  - **AxWindowsMediaPlayer:**
    - **Nombre (Name):** `wmp` (se asume).
    - **Propiedad `URL`:** Esta propiedad crucial se establece con la ruta del archivo multimedia (audio o video) que se desea cargar y reproducir. El control `AxWindowsMediaPlayer` es lo suficientemente inteligente como para manejar la reproducción de ambos tipos de archivos una vez que se le proporciona la URL.
    - **Utilización:** Es el control principal que incrusta el reproductor de Windows Media Player en el formulario y maneja la reproducción de audio y video.
  - **OpenFileDialog:**
    - **Nombre (Name):** `OpenFileDialog1`.

        - **Propiedad `Filter`:** Esta propiedad se modifica dinámicamente en los eventos `Click` de los botones para mostrar solo los tipos de archivos relevantes (audio o video). Por ejemplo, "Archivos de audio (*.mp3;*.wav)| *.mp3;*.wav".
        - **Método `ShowDialog()`:** Abre el cuadro de diálogo estándar de Windows para seleccionar un archivo.
        - **Propiedad `FileName`:** Contiene la ruta completa del archivo que el usuario ha seleccionado.
        - **Utilización:** Proporciona una interfaz gráfica para que el usuario busque y seleccione archivos de audio o video, aplicando filtros según la acción deseada.
- **Explicación de la Lógica:**
    - En el evento `btnCargarAudio_Click`:
        - La propiedad `OpenFileDialog1.Filter` se establece para que solo muestre archivos con extensiones MP3 y WAV.
        - Si el usuario selecciona un archivo (`DialogResult.OK`), `wmp.URL` se establece con la ruta del archivo elegido, cargando el audio en el reproductor.
    - En el evento `btnCargarVideo_Click`:
        - La propiedad `OpenFileDialog1.Filter` se cambia para mostrar solo archivos con extensiones MP4 y AVI.
        - De manera similar, si el usuario selecciona un archivo (`DialogResult.OK`), `wmp.URL` se establece con la ruta del archivo, cargando el video en el reproductor.
    - El control `AxWindowsMediaPlayer` se encarga de reproducir automáticamente el tipo de archivo cargado (audio o video) una vez que su propiedad `URL` se ha establecido.

Programa 156. Aplicación que controla volumen del sistema

- **Explicación del Ejercicio:** Este programa muestra conceptualmente cómo se interactuaría con el volumen del sistema utilizando un `TrackBar` para ajustar un valor, aunque la implementación real para controlar el volumen global del sistema operativo es más compleja y requiere llamadas avanzadas a la API del sistema.
- **Controles Necesarios:**
    - **TrackBar:**
        - **Nombre (Name):** `trkVolumen`.
        - **Propiedad `Minimum`:** 0. Establece el valor más bajo que el deslizador puede representar.
        - **Propiedad `Maximum`:** 100. Establece el valor más alto que el deslizador puede representar.
        - **Propiedad `Value`:** Contiene el valor numérico actual que ha sido seleccionado por el deslizador. Este valor se utiliza para actualizar la etiqueta `lblVolumen`.

        - **Evento `Scroll`:** `trkVolumen_Scroll`. Se dispara continuamente a medida que el usuario mueve el deslizador. Contiene la lógica para actualizar la etiqueta que muestra el valor del volumen.
        - **Utilización:** Permite al usuario seleccionar visualmente un valor que simula el nivel de volumen, moviendo un deslizador.
    - **Label:**
        - **Nombre (Name):** `lblVolumen`.
        - **Propiedad `Text`:** Muestra el valor numérico actual del `TrackBar` (ej., "Volumen: 50"). Se actualiza en tiempo real a medida que el usuario arrastra el deslizador.
        - **Utilización:** Informa al usuario sobre el nivel de volumen que ha seleccionado en el `TrackBar`.
- **Importación de Namespace:**
    - **`Imports System.Runtime.InteropServices`:** Es necesario para permitir la interoperabilidad con funciones nativas de la API de Windows (DLLs), aunque en este ejemplo conceptual no se utiliza directamente para el control del volumen.
- **Explicación de la Lógica:**
    - Se declara una función `SendMessageW` con el atributo `DllImport`, que es una función genérica de Windows API para enviar mensajes a ventanas. En este programa conceptual, esta función se declara pero no se implementa su uso directo para cambiar el volumen del sistema.
    - En el evento `trkVolumen_Scroll`:
        - `lblVolumen.Text = "Volumen: " & trkVolumen.Value` simplemente actualiza el texto de la etiqueta `lblVolumen` para mostrar el valor numérico actual que el usuario ha seleccionado en el `TrackBar`.
    - **Nota:** Como se indica en la fuente, el control real del volumen global del sistema es una tarea más compleja que este ejemplo no implementa directamente. Requeriría el uso de APIs de sistema avanzadas o bibliotecas externas específicas para interactuar con los componentes de audio del sistema operativo.

Programa 157. Aplicación que controla brillo de pantalla

- **Explicación del Ejercicio:** Similar al control de volumen, este programa muestra el concepto de ajustar el brillo de la pantalla. Un `TrackBar` permite al usuario seleccionar un valor, que luego se muestra en una etiqueta. Sin embargo, la implementación directa para cambiar el brillo real de la pantalla es compleja y requeriría el uso de tecnologías avanzadas del sistema operativo.
- **Controles Necesarios:**
    - **TrackBar:**
        - **Nombre (Name):** `trkBrillo`.
        - **Propiedad `Minimum`:** 0. Establece el valor más bajo que el deslizador puede representar.
        - **Propiedad `Maximum`:** 100. Establece el valor más alto que el deslizador puede representar.

        - **Propiedad `Value`:** Contiene el valor numérico actual que ha sido seleccionado por el deslizador. Este valor se utiliza para actualizar la etiqueta `lblBrillo`.
        - **Evento `Scroll`:** `trkBrillo_Scroll`. Se dispara continuamente a medida que el usuario mueve el deslizador. Contiene la lógica para actualizar la etiqueta que muestra el valor del brillo.
        - **Utilización:** Permite al usuario seleccionar visualmente un valor que simula un nivel de brillo moviendo un deslizador.
    - **Label:**
        - **Nombre (Name):** `lblBrillo`.
        - **Propiedad `Text`:** Muestra el valor numérico actual del `TrackBar` (ej., "Brillo: 50"). Se actualiza en tiempo real a medida que el usuario arrastra el deslizador.
        - **Utilización:** Informa al usuario sobre el nivel de brillo que ha seleccionado en el `TrackBar`.
- **Explicación de la Lógica:**
    - En el evento `trkBrillo_Scroll`:
        - `lblBrillo.Text = "Brillo: " & trkBrillo.Value` simplemente actualiza el texto de la etiqueta `lblBrillo` para mostrar el valor numérico actual que el usuario ha seleccionado en el `TrackBar`.
    - **Nota:** Como se indica en la fuente, ajustar el brillo real de la pantalla del sistema operativo es una tarea compleja que este ejemplo no implementa directamente. Requeriría el uso de tecnologías como WMI (Windows Management Instrumentation) o APIs específicas del sistema para interactuar con el controlador de pantalla.

Programa 158. Aplicación que toma capturas de pantalla

- **Explicación del Ejercicio:** Este programa permite al usuario tomar una captura de pantalla de la pantalla principal completa y luego la muestra en un control `PictureBox` dentro del formulario.
- **Controles Necesarios:**
    - **Button:**
        - **Nombre (Name):** `btnCaptura`.
        - **Propiedad `Text`:** "Tomar Captura".
        - **Evento `Click`:** `btnCaptura_Click`. Se activa cuando el usuario hace clic en el botón. Contiene la lógica para capturar el contenido de la pantalla principal y asignarlo al `PictureBox`.
        - **Utilización:** Dispara la acción de tomar una captura de pantalla del escritorio.
    - **PictureBox:**
        - **Nombre (Name):** `picPantalla`.
        - **Propiedad `Image`:** Almacena el objeto `Bitmap` (la imagen de la captura de pantalla) que se muestra en el control.
        - **Utilización:** Muestra la captura de pantalla que se ha realizado.
- **Explicación de la Lógica:**

- En el evento `btnCaptura_Click`:
    - `Dim bmp As New Bitmap(Screen.PrimaryScreen.Bounds.Width, Screen.PrimaryScreen.Bounds.Height)` crea un nuevo objeto `Bitmap`. Sus dimensiones se establecen para que coincidan con el ancho y alto de la pantalla principal (`Screen.PrimaryScreen.Bounds.Width` y `Height`), asegurando que el `Bitmap` tenga el tamaño exacto de la pantalla.
    - `Dim g As Graphics = Graphics.FromImage(bmp)` obtiene un objeto `Graphics` a partir del `Bitmap` recién creado. Este objeto `Graphics` es como un "lienzo" en el que se puede dibujar o pegar contenido.
    - `g.CopyFromScreen(0, 0, 0, 0, bmp.Size)` es el método clave. Copia una porción de la pantalla a la superficie del `Bitmap`. Los primeros `0, 0` son las coordenadas X e Y del punto de origen en la pantalla desde donde se iniciará la copia. Los segundos `0, 0` son las coordenadas X e Y del punto de destino en el `Bitmap` donde se pegará la copia. `bmp.Size` especifica el tamaño del área que se va a copiar, que en este caso es toda la pantalla.
    - Finalmente, `picPantalla.Image = bmp` asigna el objeto `Bitmap` (que ahora contiene la captura de pantalla) a la propiedad `Image` del `PictureBox`, lo que hace que la captura se muestre visualmente en el formulario.

Programa 159. Aplicación que graba sonido

- **Explicación del Ejercicio:** Este programa permite al usuario grabar audio desde un micrófono conectado al sistema y guardar el audio capturado en un archivo de formato WAV. Requiere el uso de una librería externa para la gestión de audio.
- **Controles Necesarios:**
    - **Button:**
        - **Nombre (Name):** `btnGrabar`.
        - **Propiedad `Text`:** "Grabar".
        - **Evento `Click`:** `btnGrabar_Click`. Se activa cuando el usuario pulsa el botón. Contiene la lógica para inicializar la grabación de audio, configurar el formato, asociar un manejador de eventos para los datos de audio, y abrir un `SaveFileDialog` para guardar el archivo WAV.
        - **Utilización:** Inicia el proceso de grabación de sonido desde el micrófono.
    - **Button:**
        - **Nombre (Name):** `btnDetener`.
        - **Propiedad `Text`:** "Detener".
        - **Evento `Click`:** `btnDetener_Click`. Se activa cuando el usuario pulsa el botón. Contiene la lógica para detener la grabación de audio en curso y cerrar el archivo de escritura, liberando los recursos.

        - **Utilización:** Detiene la grabación de sonido y finaliza el guardado del archivo.
    - **SaveFileDialog:**
        - **Nombre (Name):** `SaveFileDialog1`.
        - **Propiedad `Filter`:** Se establece para sugerir y filtrar archivos de tipo WAV (ej., "Archivos WAV (*.wav)*|.wav").
        - **Método `ShowDialog()`:** Abre el cuadro de diálogo estándar de Windows que permite al usuario especificar la ubicación y el nombre del archivo donde se guardará el audio grabado.
        - **Propiedad `FileName`:** Contiene la ruta completa (incluyendo el nombre del archivo) donde el usuario ha decidido guardar el audio.
        - **Utilización:** Permite al usuario especificar la ubicación y el nombre del archivo WAV de salida.
- **Requisito Adicional:**
    - Este programa **requiere la librería externa NAudio**. Esta librería se instala generalmente a través del administrador de paquetes NuGet en Visual Studio.
- **Importación de Namespace:**
    - **`Imports NAudio.Wave`:** Es necesario para utilizar las clases de NAudio relacionadas con la captura y escritura de audio.
- **Explicación de la Lógica:**
    - Se declaran a nivel de clase las variables `waveIn` (`WaveInEvent`, para la entrada de audio del micrófono) y `writer` (`WaveFileWriter`, para escribir los datos de audio en un archivo WAV).
    - En el evento `btnGrabar_Click`:
        - `waveIn` se inicializa como una nueva instancia de `WaveInEvent` y se configura su formato de onda (ej., 44.1 kHz de frecuencia de muestreo, mono).
        - `AddHandler waveIn.DataAvailable, Sub(s, a) writer.Write(a.Buffer, 0, a.BytesRecorded)` adjunta un manejador al evento `DataAvailable` de `waveIn`. Este evento se dispara cada vez que hay nuevos datos de audio disponibles desde el micrófono. El manejador escribe estos datos (`a.Buffer`) en el objeto `writer`.
        - Se configura `SaveFileDialog1.Filter` para archivos WAV. Si el usuario selecciona una ubicación y nombre de archivo (`DialogResult.OK`), `writer` se inicializa con el `SaveFileDialog1.FileName` y el formato de onda (`waveIn.WaveFormat`).
        - Finalmente, `waveIn.StartRecording()` inicia la captura de audio del micrófono.
    - En el evento `btnDetener_Click`:
        - `waveIn.StopRecording()` detiene la captura de audio del micrófono.
        - `writer.Close()` finaliza la escritura en el archivo WAV, asegura que todos los datos buffered se guarden y libera los recursos del archivo.

Programa 160. Aplicación que graba video

- **Explicación del Ejercicio:** Este programa es un ejemplo conceptual que demuestra cómo se podría iniciar la captura de video desde una cámara web conectada al sistema y mostrar el flujo de video en tiempo real en un `PictureBox`. La funcionalidad de grabar este video a un archivo es significativamente más compleja y no está completamente implementada en este ejemplo conceptual.
- **Controles Necesarios:**
    - **Button:**
        - **Nombre (Name):** `btnGrabar`.
        - **Propiedad `Text`:** "Grabar Video".
        - **Evento `Click`:** `btnGrabar_Click`. Se activa cuando el usuario hace clic. Contiene la lógica para detectar las cámaras web disponibles, seleccionar una (la primera por defecto), configurar un manejador de eventos para procesar los nuevos fotogramas y comenzar la captura de video.
        - **Utilización:** Inicia la captura y visualización en tiempo real del video de la cámara web.
    - **Button:**
        - **Nombre (Name):** `btnDetener`.
        - **Propiedad `Text`:** "Detener".
        - **Evento `Click`:** `btnDetener_Click`. Se activa cuando el usuario hace clic. Contiene la lógica para enviar una señal de detención a la fuente de video, si la captura está activa.
        - **Utilización:** Detiene la captura y visualización del video de la cámara web.
    - **PictureBox:**
        - **Nombre (Name):** `picVideo`.
        - **Propiedad `Image`:** Se actualiza continuamente con cada nuevo fotograma (`Bitmap`) que la cámara web captura. Esto crea la ilusión de un video en vivo.
        - **Utilización:** Muestra el flujo de video en tiempo real capturado desde la cámara web.
- **Requisito Adicional:**
    - Este programa **requiere librerías externas** como `AForge.Video` y `AForge.Video.DirectShow`. Estas se instalan generalmente a través del administrador de paquetes NuGet en Visual Studio.
- **Importación de Namespace:**
    - **`Imports AForge.Video`** y **`Imports AForge.Video.DirectShow`**: Son necesarios para utilizar las clases de AForge.NET para la captura de video.
- **Explicación de la Lógica:**
    - Se declara a nivel de clase la variable `videoSource` (`VideoCaptureDevice`), que representa el dispositivo de captura de video (la cámara web).
    - En el evento `btnGrabar_Click`:
        - `Dim videoDevices As New FilterInfoCollection(FilterCategory.VideoInputDevice`

`)` enumera todos los dispositivos de entrada de video (cámaras web) disponibles en el sistema.
    - `videoSource = New VideoCaptureDevice(videoDevices(0).MonikerString)` selecciona la primera cámara web detectada (`videoDevices(0)`) y la inicializa como la fuente de video.
    - `AddHandler videoSource.NewFrame, Sub(s, a) picVideo.Image = DirectCast(a.Clone(), Bitmap)` adjunta un manejador al evento `NewFrame` de `videoSource`. Este evento se dispara cada vez que la cámara web captura un nuevo fotograma. El manejador convierte el `EventArgs` (`a`) a un `Bitmap` y lo asigna a `picVideo.Image`, lo que hace que el fotograma se muestre en el `PictureBox`.
    - `videoSource.Start()` inicia la captura de video de la cámara web.
- En el evento `btnDetener_Click`:
    - Se verifica si `videoSource` no es `Nothing` y si está actualmente ejecutándose (`videoSource.IsRunning`).
    - Si la condición es verdadera, `videoSource.SignalToStop()` envía una señal para detener la captura de video de manera segura.

Programa 161. Aplicación que reproduce listas de reproducción

- **En qué consiste el ejercicio:** Este programa permite al usuario **crear una lista de reproducción de archivos de audio y reproducirlos** de forma secuencial o seleccionada, utilizando el control `AxWindowsMediaPlayer`. El usuario puede añadir canciones a una lista y luego reproducir la que elija.
- **Controles y propiedades:**
    - **`ListBox`**:
        - `Name`: **`lstPlaylist`**
        - **Utilización:** Este control se usa para **mostrar visualmente la lista de canciones** que el usuario ha agregado a su cola de reproducción. Cada elemento en el `ListBox` representa la ruta completa de un archivo de audio.
        - **Propiedad `Items.Add(item)`:** Se utiliza para **añadir la ruta completa** (la propiedad `FileName` del `OpenFileDialog`) de un archivo de audio seleccionado por el usuario a la colección de elementos de la lista de reproducción.
        - **Propiedad `SelectedItem`:** Cuando el usuario hace clic en el botón "Reproducir", esta propiedad **obtiene la ruta de la canción actualmente resaltada o seleccionada** en el `ListBox`. Esta ruta es crucial para indicarle al reproductor qué archivo debe cargar y tocar.
    - **`Button`**:
        - `Name`: **`btnAgregar`**, `Text`: "Agregar Canción"

        - **Utilización:** Es el botón que el usuario presiona para **iniciar el proceso de selección de archivos de audio** desde su sistema de archivos. Al hacer clic, se abre un diálogo que permite al usuario buscar y seleccionar canciones.
        - `Name`: **`btnReproducir`**, `Text`: "Reproducir"
        - **Utilización:** Este botón, cuando se presiona, **activa la reproducción del archivo de audio** que el usuario haya seleccionado previamente en el `lstPlaylist`. Se asegura de que haya un elemento seleccionado antes de intentar reproducir.
    - **`OpenFileDialog`**:
        - `Name`: **`OpenFileDialog1`**
        - **Propiedad `Filter`:** Se configura con el valor `"MP3|*.mp3|WAV|*.wav"`. Esto es fundamental para **limitar los tipos de archivos que el usuario puede seleccionar** en el diálogo. Solo se mostrarán archivos con extensiones `.mp3` o `.wav`, mejorando la experiencia del usuario y asegurando la compatibilidad con el reproductor.
        - **Método `ShowDialog()`:** Abre el cuadro de diálogo estándar de Windows que permite al usuario navegar por su sistema de archivos y **seleccionar uno o más archivos**. La función devuelve `DialogResult.OK` si el usuario selecciona un archivo y hace clic en "Abrir".
        - **Propiedad `FileName`:** Después de que el usuario selecciona un archivo y `ShowDialog()` devuelve `OK`, esta propiedad contiene la **ruta completa del archivo seleccionado**. Esta ruta se añade a la `lstPlaylist`.
    - **`AxWindowsMediaPlayer`**:
        - `Name`: **`wmp`**
        - **Utilización:** Es un control de reproductor multimedia incrustado que permite la **reproducción de archivos de audio y video**. Es el motor que realmente reproduce la música en este programa.
        - **Propiedad `URL`:** Esta propiedad se utiliza para **especificar la ruta del archivo multimedia que se va a reproducir**. Cuando el usuario selecciona una canción en el `lstPlaylist` y pulsa "Reproducir", la `URL` del `wmp` se actualiza con la ruta de esa canción.
        - **Método `Ctlcontrols.play()`:** Este método **inicia la reproducción del archivo multimedia** que está actualmente cargado en el reproductor. Se llama cuando el usuario hace clic en el botón `btnReproducir`.

Programa 162. Aplicación que descarga archivos de internet

- **En qué consiste el ejercicio:** Este programa permite al usuario **descargar un archivo desde una URL (dirección web) específica** de internet y guardarlo en una ubicación local de su elección en el disco duro.
- **Controles y propiedades:**
    - **`TextBox`**:
        - `Name`: **`txtURL`**

        - **Utilización:** Sirve como un campo de entrada donde el usuario debe **escribir o pegar la URL completa del archivo que desea descargar**.
    - **`Button`**:
        - `Name`: **`btnDescargar`**, `Text`: "Descargar"
        - **Utilización:** Al hacer clic en este botón, se **inicia el proceso de descarga**. El programa toma la URL del `txtURL` y, después de que el usuario especifica dónde guardar el archivo, procede a bajarlo.
    - **`SaveFileDialog`**:
        - `Name`: **`SaveFileDialog1`**
        - **Método `ShowDialog()`:** Abre el cuadro de diálogo estándar de Windows que permite al usuario **seleccionar la carpeta de destino y el nombre del archivo** con el que se guardará el contenido descargado. Si el usuario confirma la operación (haciendo clic en "Guardar"), el método devuelve `DialogResult.OK`.
        - **Propiedad `FileName`:** Una vez que el usuario ha seleccionado una ruta y un nombre en el diálogo, esta propiedad **contiene la ruta completa y el nombre del archivo local** donde el archivo descargado será guardado.
    - **`WebClient`**: (Clase del espacio de nombres `System.Net`)
        - **Utilización:** Es una clase de .NET que **facilita la interacción con recursos web**, incluyendo la descarga de datos. Se instancia un objeto de esta clase para realizar la operación de descarga.
        - **Método `DownloadFile(address, fileName)`:** Este es el método clave para la descarga. Toma dos argumentos: la **URL de origen del archivo** (`txtURL.Text`) y la **ruta de destino local** (`SaveFileDialog1.FileName`). Una vez llamado, el `WebClient` gestiona la descarga del archivo binario.
    - **`MessageBox`**:
        - **Utilización:** Se utiliza para **mostrar un mensaje de confirmación** al usuario, informándole que el proceso de descarga se ha completado exitosamente.

Programa 163. Aplicación que sube archivos a servidor FTP

- **En qué consiste el ejercicio:** Este programa permite al usuario **seleccionar un archivo de su disco local y subirlo a un servidor FTP** (File Transfer Protocol). Para ello, requiere que el usuario introduzca la dirección del servidor y sus credenciales de autenticación (usuario y contraseña).
- **Controles y propiedades:**
    - **`TextBox`**:
        - `Name`: **`txtServidor`**, `Text`: "ftp://servidor.com/"
        - **Utilización:** Campo de texto donde el usuario **especifica la URL base del servidor FTP** al que se conectará para subir el archivo. Es importante que la URL incluya el esquema "ftp://".
        - `Name`: **`txtUsuario`**

    - **Utilización:** Campo para que el usuario **introduzca el nombre de usuario** necesario para autenticarse en el servidor FTP. Sin las credenciales correctas, la subida no será autorizada.
    - `Name`: **`txtPassword`**
    - **Utilización:** Campo para que el usuario **introduzca la contraseña** asociada al nombre de usuario del FTP. Este campo generalmente se configura con `PasswordChar` para ocultar la entrada.
- **`Button`**:
    - `Name`: **`btnSubir`**, `Text`: "Subir Archivo"
    - **Utilización:** Cuando el usuario hace clic en este botón, **se inicia el flujo de selección del archivo local y su posterior subida** al servidor FTP.
- **`OpenFileDialog`**:
    - `Name`: **`OpenFileDialog1`**
    - **Método `ShowDialog()`:** Abre el diálogo estándar de Windows para que el usuario **seleccione el archivo local que desea subir**. Devuelve `DialogResult.OK` si el usuario elige un archivo y hace clic en "Abrir".
    - **Propiedad `FileName`:** Contiene la **ruta completa del archivo local** que ha sido seleccionado por el usuario. Esta ruta se utiliza para leer el contenido del archivo y para construir la URL de destino en el servidor FTP.
- **`FtpWebRequest`**: (Clase del espacio de nombres `System.Net`)
    - **Utilización:** Es la clase principal para **interactuar con servidores FTP**. Permite crear solicitudes para operaciones FTP específicas como subir, descargar o listar archivos.
    - **Método `WebRequest.Create(uri)`:** Se utiliza para crear una nueva instancia de `FtpWebRequest` a partir de una URI (Uniform Resource Identifier) que combina la URL del servidor y el nombre del archivo a subir.
    - **Propiedad `Method`:** Se establece en **`WebRequestMethods.Ftp.UploadFile`** para indicar que la operación que se va a realizar en el servidor FTP es la de cargar un archivo.
    - **Propiedad `Credentials`:** Se asigna un objeto `NetworkCredential` (`New NetworkCredential(txtUsuario.Text, txtPassword.Text)`) que **contiene las credenciales del usuario** para la autenticación en el servidor FTP. Esto es esencial para el acceso autorizado.
    - **Propiedad `ContentLength`:** Se establece con la **longitud en bytes del archivo** que se va a subir (`fileContents.Length`). Informar al servidor de la longitud del contenido ayuda en el proceso de transferencia.
    - **Método `GetRequestStream()`:** Obtiene un **flujo de datos (`requestStream`) al cual se deben escribir los bytes del archivo local**. Este flujo es el canal por el cual los datos del archivo se envían al servidor FTP.
- **`NetworkCredential`**: (Clase del espacio de nombres `System.Net`)
    - **Utilización:** Encapsula las **credenciales (nombre de usuario y contraseña)** que se utilizan para autenticar una solicitud de red, en este caso, con el servidor FTP.

- **`IO.File.ReadAllBytes(path)`**: (Método de `System.IO`)
    - **Utilización:** Lee el **contenido completo del archivo local seleccionado** en una matriz de bytes. Esta matriz de bytes es lo que se envía al servidor FTP.
- **`requestStream.Write(buffer, offset, count)`**:
    - **Utilización:** Escribe la matriz de bytes (`fileContents`) que contiene el contenido del archivo en el `requestStream`, **enviando los datos al servidor FTP**.
- **`requestStream.Close()`**:
    - **Utilización:** Cierra el flujo de solicitud, lo cual es importante para **finalizar la operación de subida y liberar los recursos** de red asociados.
- **`MessageBox`**:
    - **Utilización:** Muestra un mensaje de confirmación para el usuario, indicando que el archivo ha sido subido correctamente al servidor FTP.

Programa 164. Aplicación que se conecta a una API REST

- **En qué consiste el ejercicio:** Este programa demuestra cómo realizar una **solicitud GET básica a una API REST (Application Programming Interface Representational State Transfer)** y cómo obtener y mostrar la respuesta de esa API en un control de texto. Es un ejemplo fundamental de cómo una aplicación puede interactuar con servicios web modernos.
- **Controles y propiedades:**
    - **`TextBox`**:
        - `Name`: **`txtURL`**
        - **Utilización:** Es el campo donde el usuario **introduce la URL del endpoint (punto de acceso) de la API REST** a la que desea realizar la consulta. Esta URL define el recurso específico al que se accede.
        - `Name`: **`txtResultado`**, `Multiline`: `True`, `ScrollBars`: `Vertical`
        - **Utilización:** Un control de texto multilínea diseñado para **mostrar la respuesta completa recibida de la API REST**. Las propiedades `Multiline` y `ScrollBars` son útiles para visualizar respuestas largas o con formato de texto estructurado como JSON o XML.
    - **`Button`**:
        - `Name`: **`btnConsultar`**, `Text`: "Consultar"
        - **Utilización:** Al hacer clic en este botón, se **activa el proceso de conexión a la API REST y la descarga de su respuesta** desde la URL especificada en `txtURL`.
    - **`WebClient`**: (Clase del espacio de nombres `System.Net`)
        - **Utilización:** Es una clase de .NET que **simplifica las tareas de comunicación con recursos web**, como la descarga de datos desde una URL. Se crea una nueva instancia para cada operación.
        - **Método `DownloadString(address)`:** Este es el método clave del programa. Toma la URL del `txtURL.Text` y **descarga el contenido de ese**

recurso web como una cadena de texto. Esta cadena es la respuesta (en formato JSON, XML, etc.) que la API REST devuelve. El resultado de esta operación se asigna directamente a la propiedad `Text` de `txtResultado`.

Programa 165. Aplicación que se conecta a una API SOAP

- **En qué consiste el ejercicio:** Este programa es un **ejemplo conceptual** que ilustra cómo se interactuaría con una **API SOAP (Simple Object Access Protocol)**. A diferencia de las API REST, las API SOAP a menudo requieren la generación de un "cliente de servicio" en el proyecto, que encapsula la complejidad de la comunicación. El código muestra cómo se invocaría un método de este cliente generado y se mostraría su resultado.
- **Controles y propiedades:**
  - **`TextBox`**:
    - `Name`: **`txtServicio`**
    - **Utilización:** En una aplicación real, este `TextBox` podría usarse para **especificar la URL (WSDL) del servicio SOAP**, aunque en el código de ejemplo no se interactúa directamente con su contenido para la llamada al servicio.
    - `Name`: **`txtResultado`**, `Multiline`: `True`, `ScrollBars`: `Vertical`
    - **Utilización:** Un control de texto multilínea que está diseñado para **mostrar el resultado que se obtiene al llamar a un método específico del servicio SOAP**. Si el servicio devuelve una cadena o un valor que puede convertirse a cadena, se mostrará aquí.
  - **`Button`**:
    - `Name`: **`btnConsultar`**, `Text`: "Consultar"
    - **Utilización:** Al hacer clic en este botón, se **activa la lógica conceptual para invocar un método del cliente del servicio SOAP** y luego se muestra el resultado obtenido en el `txtResultado`.
  - **`ServiceReference1.MiServicioSoapClient`**: (Clase generada automáticamente)
    - **Utilización:** Esta es una clase de cliente que **Visual Studio genera automáticamente** cuando se añade una "Referencia de Servicio" (o "Referencia Web" en proyectos más antiguos) a un servicio SOAP existente. Esta clase se encarga de gestionar los detalles de la comunicación SOAP y expone los métodos del servicio como si fueran métodos locales de la aplicación. El nombre `ServiceReference1` sería el nombre que se le da a la referencia en el proyecto, y `MiServicioSoapClient` un nombre de clase derivado del servicio.
    - **Método `MiMetodo("parametro")`:** Representa una **llamada a una función específica que el servicio SOAP expone**. En este ejemplo conceptual, se asume que existe un método llamado `MiMetodo` que toma una cadena como parámetro ("parametro") y devuelve un resultado que se asigna a `txtResultado.Text`. Los nombres de los métodos y sus

parámetros dependerán completamente de la definición del servicio SOAP real.

Programa 166. Aplicación que consume servicios web JSON

- **En qué consiste el ejercicio:** Este programa establece una conexión con un servicio web, descarga datos que están formateados en **JSON (JavaScript Object Notation)**, y luego utiliza una librería externa para analizar (parsear) estos datos JSON y mostrarlos en un cuadro de texto. Es un ejemplo común de cómo interactuar con muchas APIs web modernas.
- **Controles y propiedades:**
  - **`TextBox`**:
    - `Name`: **`txtURL`**
    - **Utilización:** Un campo de entrada donde el usuario **especifica la URL del servicio web** que se espera que devuelva datos en formato JSON.
    - `Name`: **`txtResultado`**, `Multiline`: `True`
    - **Utilización:** Un control de texto multilínea que **muestra el contenido JSON que se ha descargado y procesado**. Es útil para visualizar la estructura y los datos recibidos del servicio web.
  - **`Button`**:
    - `Name`: **`btnConsultar`**, `Text`: "Consultar"
    - **Utilización:** Al hacer clic en este botón, se **inicia el proceso de conexión al servicio web, descarga los datos JSON y los procesa** para su visualización.
  - **`WebClient`**: (Clase del espacio de nombres `System.Net`)
    - **Utilización:** Una clase que simplifica la interacción con recursos web. Se usa para **descargar la cadena de texto JSON** desde la URL especificada.
    - **Método `DownloadString(address)`:** Descarga todo el contenido de la URL (`txtURL.Text`) como una única cadena de texto. En este caso, esta cadena es el documento JSON.
  - **`JObject`**: (Clase del espacio de nombres `Newtonsoft.Json.Linq`)
    - **Utilización:** Esta clase es parte de la librería externa `Newtonsoft.Json` (también conocida como Json.NET), que es una herramienta muy popular para **trabajar con JSON en aplicaciones .NET**. Proporciona métodos para analizar y manipular objetos JSON.
    - **Método `Parse(jsonString)`:** Toma una cadena de texto JSON (`json`) y la **analiza, convirtiéndola en un objeto `JObject`** en memoria. Este objeto representa la estructura jerárquica de los datos JSON, lo que permite un acceso más fácil a sus elementos.
  - **`obj.ToString()`**:
    - **Utilización:** Después de que la cadena JSON se ha parseado en un `JObject`, este método convierte el objeto `JObject` de nuevo a una **cadena formateada**. Esto es útil para mostrar el JSON en un formato legible en el `txtResultado`.

Programa 167. Aplicación que consume servicios web XML

- **En qué consiste el ejercicio:** Este programa se conecta a un servicio web, **descarga datos que están formateados en XML (Extensible Markup Language)**, los carga en un modelo de documento XML en memoria y luego muestra el contenido completo del documento XML en un cuadro de texto.
- **Controles y propiedades:**
    - **`TextBox`**:
        - `Name`: **`txtURL`**
        - **Utilización:** Un campo de entrada donde el usuario **ingresa la URL del servicio web** que se espera que devuelva datos en formato XML.
        - `Name`: **`txtResultado`**, `Multiline`: `True`
        - **Utilización:** Un control de texto multilínea que se usa para **mostrar el contenido XML completo** después de haber sido descargado y cargado en un `XmlDocument`. Es útil para la depuración y visualización de la estructura de los datos XML.
    - **`Button`**:
        - `Name`: **`btnConsultar`**, `Text`: "Consultar"
        - **Utilización:** Al hacer clic en este botón, se **inicia la conexión al servicio web, la descarga de los datos XML y su posterior procesamiento** para mostrar el resultado en `txtResultado`.
    - **`WebClient`**: (Clase del espacio de nombres `System.Net`)
        - **Utilización:** Una clase que simplifica las operaciones de red, como la descarga de datos desde una URL. Se usa para **descargar la cadena de texto XML** del servicio web.
        - **Método `DownloadString(address)`:** Descarga el contenido completo de la URL (`txtURL.Text`) como una cadena de texto. Esta cadena es el documento XML que se procesará.
    - **`XmlDocument`**: (Clase del espacio de nombres `System.Xml`)
        - **Utilización:** Proporciona un modelo en memoria para documentos XML, permitiendo su carga, manipulación y guardado. Es la clase central para trabajar con XML en este programa.
        - **Método `LoadXml(xmlString)`:** Toma una cadena de texto XML (`xmlString`) y la **carga en el objeto `XmlDocument`**, parseando su estructura y permitiendo el acceso a sus nodos y atributos.
        - **Propiedad `OuterXml`:** Devuelve la **representación en cadena del documento XML completo**, incluyendo todas las etiquetas, atributos y contenido, desde el nodo raíz hasta el final. Esta es la cadena que se muestra en `txtResultado`.

Programa 168. Cliente de chat local simple

- **En qué consiste el ejercicio:** Este programa simula un cliente de chat básico de forma local. Los mensajes que el usuario escribe se añaden a un `ListBox`, creando una representación

visual de una conversación. **Es importante destacar que no hay comunicación real por red**; los mensajes solo se gestionan y muestran dentro de la misma aplicación.

- **Controles y propiedades:**
    - **`TextBox`**:
        - `Name`: **`txtMensaje`**
        - **Utilización:** Es el campo de entrada donde el usuario **escribe el mensaje que desea "enviar"**.
        - **Método `Clear()`:** Después de que el mensaje se añade al `lstChat`, este método **vacía el contenido del `TextBox`**, dejándolo listo para que el usuario escriba un nuevo mensaje.
    - **`Button`**:
        - `Name`: **`btnEnviar`**, `Text`: "Enviar"
        - **Utilización:** Cuando el usuario hace clic en este botón, el texto que está en el `txtMensaje` se procesa y se añade a la lista de mensajes del chat. Simula la acción de enviar un mensaje en un chat real.
    - **`ListBox`**:
        - `Name`: **`lstChat`**
        - **Utilización:** Este control es el área principal donde se **muestran todos los mensajes de la conversación**, ordenados cronológicamente a medida que se "envían". Actúa como el historial visible del chat.
        - **Propiedad `Items.Add(item)`:** Se utiliza para **añadir cada nuevo mensaje** (que se construye concatenando "Tú: " con el texto del `txtMensaje`) a la colección de elementos que se muestran en el `ListBox`.

Programa 169. Cliente de chat en red local

- **En qué consiste el ejercicio:** Este programa es un **ejemplo conceptual** de cómo un cliente de chat podría enviar mensajes a través de una red local. Utiliza las clases `TcpClient` y `NetworkStream` para demostrar la mecánica de la comunicación en red. Sin embargo, para que funcione realmente, **requeriría una implementación de servidor de chat separada** y una conexión activa a ese servidor, lo cual no se proporciona en este código.
- **Controles y propiedades:**
    - **`TextBox`**:
        - `Name`: **`txtMensaje`**
        - **Utilización:** Campo donde el usuario **escribe el mensaje que desea enviar** a través de la red local.
        - **Método `Clear()`:** Después de que el mensaje es enviado, **vacía el `TextBox`** para que el usuario pueda escribir una nueva entrada.
    - **`Button`**:
        - `Name`: **`btnEnviar`**, `Text`: "Enviar"

        - **Utilización:** Al hacer clic en este botón, se **inicia el proceso de envío del mensaje**. El texto se convierte a bytes y se escribe en el flujo de red para su transmisión, y también se añade al `lstChat` local.
    - **`ListBox`**:
        - `Name`: **`lstChat`**
        - **Utilización:** Muestra los **mensajes que el usuario ha "enviado"**, actuando como un historial visual de las interacciones locales del cliente de chat.
        - **Propiedad `Items.Add(item)`:** Añade el mensaje (precedido por "Tú: ") a la lista visible de mensajes.
    - **`TcpClient`**: (Clase del espacio de nombres `System.Net.Sockets`)
        - **Utilización:** Es la clase que proporciona **servicios de conexión para clientes TCP**. Se usaría para establecer una conexión con un servidor de chat remoto o local. En este ejemplo, la variable `client` se declara, pero su inicialización y conexión real no están en el fragmento de código.
    - **`NetworkStream`**: (Clase del espacio de nombres `System.Net.Sockets`)
        - **Utilización:** Representa un **flujo de datos en la red subyacente** (como una conexión `TcpClient`). Una vez que se establece una conexión, se obtiene un `NetworkStream` para enviar y recibir datos. La variable `stream` se declara, pero su inicialización no se muestra.
        - **Método `Write(buffer, offset, size)`:** Este método se utiliza para **escribir una secuencia de bytes (`msg`) en el flujo de red**, enviándolos al destino (el servidor de chat). Los parámetros especifican la matriz de bytes, el desplazamiento inicial y la cantidad de bytes a escribir.
    - **`Encoding.ASCII.GetBytes(string)`**: (Método de la clase `Encoding` en `System.Text`)
        - **Utilización:** Se encarga de **convertir una cadena de texto (el mensaje del `txtMensaje`) en una matriz de bytes** utilizando la codificación ASCII. Esta conversión es necesaria porque las comunicaciones de red operan con bytes, no directamente con cadenas de texto.

Programa 170. Cliente de chat con base de datos

- **En qué consiste el ejercicio:** Este programa implementa un cliente de chat que no solo muestra los mensajes localmente, sino que también los **almacena de forma persistente en una base de datos SQLite**. Esto significa que los mensajes no se perderán cuando la aplicación se cierre y se podrán recuperar más tarde. El programa también se encarga de crear la tabla de mensajes si no existe.
- **Controles y propiedades:**
    - **`TextBox`**:
        - `Name`: **`txtMensaje`**
        - **Utilización:** Es el campo de entrada donde el usuario **escribe el mensaje que desea enviar** al chat y que será guardado en la base de datos.

    - **Método `Clear()`:** Después de enviar el mensaje, este método **vacía el contenido del `TextBox`**, preparándolo para el próximo mensaje.
- **`Button`**:
    - `Name`: **`btnEnviar`**, `Text`: "Enviar"
    - **Utilización:** Al hacer clic en este botón, **el mensaje se guarda en la base de datos SQLite y también se añade al `lstChat`** visible en el formulario.
- **`ListBox`**:
    - `Name`: **`lstChat`**
    - **Utilización:** Muestra los **mensajes de la conversación de chat** al usuario, reflejando tanto los mensajes enviados como (presumiblemente, en una implementación completa) los recibidos o cargados de la base de datos.
    - **Propiedad `Items.Add(item)`:** Se utiliza para **añadir cada nuevo mensaje** (construido con "Tú: " y el texto del `txtMensaje`) a la colección de elementos mostrados en el `ListBox`.
- **`SQLiteConnection`**: (Clase del espacio de nombres `System.Data.SQLite`)
    - **Utilización:** Representa una **conexión abierta a una base de datos SQLite**. La cadena de conexión (`"Data Source=chat.db;Version=3;"`) especifica que se conectará a un archivo de base de datos llamado `chat.db` en la versión 3 de SQLite.
    - **Método `Open()`:** Establece la conexión física con el archivo de la base de datos. Es un paso necesario antes de ejecutar cualquier comando SQL.
- **`SQLiteCommand`**: (Clase del espacio de nombres `System.Data.SQLite`)
    - **Utilización:** Se usa para **ejecutar comandos SQL** (como `CREATE TABLE` o `INSERT`) contra la base de datos SQLite. Se crea una nueva instancia para cada operación SQL.
    - **Método `AddWithValue("@parameterName", value)`:** Se utiliza para **añadir parámetros a las consultas SQL**. En el caso de la inserción de mensajes, `@msg` se mapea al texto del `txtMensaje`. Esta práctica es crucial para **prevenir ataques de inyección SQL** y para manejar correctamente los valores de cadena que pueden contener caracteres especiales.
    - **Método `ExecuteNonQuery()`:** Ejecuta una instrucción SQL que no devuelve un conjunto de resultados (filas), como las sentencias `INSERT`, `UPDATE`, `DELETE` o `CREATE TABLE`. Devuelve el número de filas afectadas por la operación.
- **`Form1_Load` (Evento del formulario):**
    - **Utilización:** Este evento se dispara cuando el formulario se carga por primera vez. Aquí, se abre la conexión a la base de datos y se ejecuta el comando `CREATE TABLE IF NOT EXISTS` para **asegurarse de que la tabla `mensajes` exista** con sus columnas `usuario` y `mensaje`. Esto garantiza que la base de datos esté lista para almacenar datos desde el inicio.

Programa 171. Cliente de chat con login de usuario

- **En qué consiste el ejercicio:** Este programa añade una **funcionalidad de inicio de sesión básica** a un cliente de chat. El usuario debe introducir un nombre de usuario y una contraseña. El sistema verifica si los campos no están vacíos y, si es así, simula un inicio de sesión exitoso mostrando un mensaje de bienvenida. Es una introducción a la gestión de accesos en aplicaciones.
- **Controles y propiedades:**
  - **`TextBox`**:
    - `Name`: **`txtUsuario`**
    - **Utilización:** Es el campo donde el usuario **ingresa su nombre de usuario** para intentar iniciar sesión.
    - `Name`: **`txtContraseña`**, `PasswordChar`: `*`
    - **Utilización:** Es el campo donde el usuario **ingresa su contraseña**. La propiedad `PasswordChar` se establece en `*` (asterisco) para que los caracteres escritos se muestren como asteriscos en lugar del texto real, proporcionando una medida básica de seguridad visual.
  - **`Button`**:
    - `Name`: **`btnLogin`**, `Text`: "Iniciar Sesión"
    - **Utilización:** Al hacer clic en este botón, se **activa la lógica de verificación de los campos de usuario y contraseña**. Si ambos campos están llenos, se simula un inicio de sesión exitoso; de lo contrario, se solicita al usuario que complete la información.
  - **`usuarioLogueado` (variable de clase `String`):**
    - **Utilización:** Una variable declarada a nivel de clase que **almacena el nombre del usuario que ha iniciado sesión exitosamente**. Se utiliza para personalizar el mensaje de bienvenida.
  - **`MessageBox`**:
    - **Utilización:** Se utiliza para **mostrar retroalimentación al usuario**:
      - Un mensaje de bienvenida (`"Bienvenido " & usuarioLogueado`) si los campos de usuario y contraseña están llenos.
      - Una alerta (`"Ingrese usuario y contraseña"`) si alguno de los campos está vacío, guiando al usuario a completar la información necesaria.
    - **Validación `If txtUsuario.Text <> "" And txtContraseña.Text <> "" Then`**:
      - **Utilización:** Esta condición verifica si **ambos campos de texto (`txtUsuario` y `txtContraseña`) no están vacíos**. Es una validación básica para asegurar que el usuario ha intentado introducir algún dato antes de proceder con el "inicio de sesión". (Es importante notar que este programa solo valida que los campos no estén vacíos, no que las credenciales sean correctas contra un usuario/contraseña predefinido o una base de datos, a diferencia de P124).

Programa 172. Cliente de chat con historial de mensajes

- **En qué consiste el ejercicio:** Este programa extiende la funcionalidad de un cliente de chat al implementar un **historial persistente de mensajes**. Al iniciar la aplicación, carga los mensajes previos desde un archivo de texto. Cada vez que el usuario envía un nuevo mensaje, este se añade tanto al chat visible como al archivo de historial, asegurando que la conversación se guarde y pueda recuperarse.
- **Controles y propiedades:**
    - **`TextBox`**:
        - `Name`: **`txtMensaje`**
        - **Utilización:** Es el campo de entrada donde el usuario **escribe el mensaje que desea enviar**. El contenido de este `TextBox` se agregará al chat y se guardará en el archivo de historial.
        - **Método `Clear()`:** Después de que el mensaje es enviado y registrado, este método **vacía el contenido del `TextBox`**, dejándolo listo para una nueva entrada.
    - **`Button`**:
        - `Name`: **`btnEnviar`**, `Text`: "Enviar"
        - **Utilización:** Al hacer clic en este botón, el programa **toma el mensaje del `txtMensaje`, lo añade al `lstChat` y también lo escribe en el archivo de historial**.
    - **`ListBox`**:
        - `Name`: **`lstChat`**
        - **Utilización:** Muestra visualmente el **historial de mensajes de la conversación**. Al inicio, se llena con el contenido del archivo de historial, y luego se actualiza con cada nuevo mensaje enviado.
        - **Propiedad `Items.AddRange(collection)`:** Se utiliza en el evento `Form1_Load` para **cargar todas las líneas del archivo de historial** (`File.ReadAllLines(historialArchivo)`) directamente en el `ListBox`, poblándolo con los mensajes previos.
        - **Propiedad `Items.Add(item)`:** Añade un nuevo mensaje (precedido por "Tú: ") a la colección de elementos visibles en el `ListBox` cuando el usuario envía un mensaje.
    - **`historialArchivo` (variable de clase `String`):**
        - **Utilización:** Define la **ruta y el nombre del archivo de texto** (`"historial.txt"`) donde se guardarán y de donde se cargarán los mensajes del chat. Se declara a nivel de clase para que sea accesible en todo el formulario.
    - **`Form1_Load` (Evento del formulario):**
        - **Utilización:** Este evento se dispara cuando el formulario se carga por primera vez. Su lógica **verifica si el `historialArchivo` existe**. Si es así, lee todas

sus líneas (`File.ReadAllLines`) y las añade al `lstChat` (`Items.AddRange`), restaurando así el historial de la conversación.

- **`File.Exists(path)`**: (Método de la clase `File` en `System.IO`)
  - **Utilización:** Se usa para **comprobar si el archivo especificado por `historialArchivo` existe** en el sistema de archivos. Esto evita errores al intentar leer un archivo que no está presente.
- **`File.ReadAllLines(path)`**: (Método de la clase `File` en `System.IO`)
  - **Utilización:** Lee **todas las líneas de texto de un archivo** y las devuelve como una matriz de cadenas. Cada elemento de la matriz corresponde a una línea del archivo. Es ideal para cargar el historial completo de una vez.
- **`File.AppendAllText(path, contents)`**: (Método de la clase `File` en `System.IO`)
  - **Utilización:** Abre un archivo, **añade el texto especificado (`"Tú: " & txtMensaje.Text & Environment.NewLine`) al final** de este, y luego lo cierra. Este método es perfecto para añadir nuevos mensajes al historial sin sobrescribir los anteriores.
- **`Environment.NewLine`**:
  - **Utilización:** Una propiedad que representa la **cadena de salto de línea específica del entorno** (por ejemplo, `CRLF` en Windows). Se usa para asegurar que cada mensaje se guarde en una línea separada en el archivo de texto, manteniendo el formato del historial.

Programa 173. Cliente de chat con emoticonos

- **En qué consiste el ejercicio:** Este programa mejora la experiencia del usuario en un cliente de chat al permitirle **insertar emoticonos predefinidos en sus mensajes**. Los emoticonos se seleccionan de un `ComboBox` y luego se adjuntan al texto que el usuario escribe, apareciendo juntos en el historial del chat.
- **Controles y propiedades:**
  - **`TextBox`**:
    - `Name`: **`txtMensaje`**
    - **Utilización:** Es el campo de entrada principal donde el usuario **escribe el contenido textual de su mensaje**. El emoticono seleccionado se añadirá al final de este texto antes de mostrarse en el chat.
    - **Método `Clear()`:** Después de que el mensaje (con el emoticono) se ha añadido al `lstChat`, este método **vacía el `TextBox`**, dejándolo listo para el siguiente mensaje.
  - **`Button`**:
    - `Name`: **`btnEnviar`**, `Text`: "Enviar"
    - **Utilización:** Al hacer clic en este botón, se **compila el mensaje combinando el texto de `txtMensaje` y el emoticono seleccionado en `cmbEmoticonos`**, y luego se añade este mensaje completo al `lstChat`.
  - **`ComboBox`**:

        - Name: **cmbEmoticonos**, Items: 😀, 😳, 😎
        - **Utilización:** Un control desplegable que ofrece una **lista de emoticonos predefinidos** para que el usuario pueda elegir uno. Los ítems (😀, 😳, 😎) se añadirían a la propiedad `Items` del `ComboBox` en el diseñador o en el código.
        - **Propiedad `SelectedItem`:** Cuando el usuario hace clic en `btnEnviar`, esta propiedad **obtiene el emoticono específico que el usuario ha elegido** de la lista desplegable. Este valor se usa para adjuntarlo al mensaje.
    - **`ListBox`**:
        - Name: **`lstChat`**
        - **Utilización:** Es el área donde se **muestran todos los mensajes de la conversación**, incluyendo tanto el texto escrito por el usuario como el emoticono que haya seleccionado.
        - **Propiedad `Items.Add(item)`:** Se utiliza para **añadir el mensaje completo** (el texto del usuario concatenado con un espacio y el emoticono) a la colección de elementos visibles en el `ListBox`.

Programa 174. Cliente de chat con archivos adjuntos

- **En qué consiste el ejercicio:** Este programa extiende la funcionalidad de un cliente de chat permitiendo al usuario **"adjuntar" un archivo local a un mensaje**. Al seleccionar un archivo, su ruta se almacena y luego se muestra junto con el texto del mensaje en el historial del chat. **Es importante destacar que en este ejemplo, solo se adjunta la *ruta* del archivo al mensaje en el chat, no el archivo binario en sí**.
- **Controles y propiedades:**
    - **`TextBox`**:
        - Name: **`txtMensaje`**
        - **Utilización:** Campo de entrada donde el usuario **escribe el texto principal de su mensaje**. Si se ha adjuntado un archivo, su ruta aparecerá junto a este texto en el `lstChat`.
        - **Método `Clear()`:** Después de enviar el mensaje, **vacía el `TextBox`** para que el usuario pueda escribir un nuevo mensaje.
    - **`Button`**:
        - Name: **`btnEnviar`**, `Text`: "Enviar"
        - **Utilización:** Al hacer clic, **crea el mensaje combinando el texto de `txtMensaje` con la ruta del archivo adjunto** (si existe), lo añade al `lstChat`, limpia `txtMensaje` y restablece la variable `archivoAdjunto`.
        - Name: **`btnAdjuntar`**, `Text`: "Adjuntar Archivo"
        - **Utilización:** Cuando el usuario hace clic aquí, se **abre el `OpenFileDialog` para que pueda seleccionar un archivo** de su sistema. La ruta del archivo seleccionado se guarda para adjuntarla al mensaje.
    - **`OpenFileDialog`**:

        - Name: **`OpenFileDialog1`**
        - **Método `ShowDialog()`:** Abre el cuadro de diálogo estándar de Windows que permite al usuario **navegar y seleccionar un archivo local**. Si el usuario selecciona un archivo y hace clic en "Abrir", devuelve `DialogResult.OK`.
        - **Propiedad `FileName`:** Si el `ShowDialog()` devuelve `OK`, esta propiedad **contiene la ruta completa del archivo que el usuario ha seleccionado**. Esta ruta se asigna a la variable `archivoAdjunto`.
    - **`ListBox`**:
        - Name: **`lstChat`**
        - **Utilización:** Muestra el **historial de mensajes de la conversación**, incluyendo el texto del usuario y la ruta del archivo que se ha "adjuntado".
        - **Propiedad `Items.Add(item)`:** Añade el mensaje completo (texto + ruta del archivo) a la colección de elementos visibles en el `ListBox`.
    - **`archivoAdjunto` (variable de clase `String`):**
        - **Utilización:** Esta variable de clase **almacena la ruta completa del archivo que el usuario ha seleccionado** mediante el `OpenFileDialog`. Su valor persiste hasta que se envía el mensaje, momento en el que se restablece a una cadena vacía para el próximo adjunto.
    - **`MessageBox`**:
        - **Utilización:** Muestra un mensaje informativo para confirmar al usuario que un archivo ha sido adjuntado y cuál es su ruta.

Programa 175. Cliente de chat con múltiples salas

- **En qué consiste el ejercicio:** Este programa introduce el concepto de **múltiples "salas" o canales de conversación** en un cliente de chat. El usuario puede seleccionar una sala de un `ComboBox`, y el chat visible (`ListBox`) se actualizará para mostrar solo los mensajes de esa sala. Cada sala mantiene su propio historial de mensajes de forma independiente.
- **Controles y propiedades:**
    - **`ComboBox`**:
        - Name: **`cmbSalas`**, `Items`: "General", "Juegos", "Música"
        - **Utilización:** Es un control desplegable que permite al usuario **elegir en qué sala de chat desea estar o qué historial desea visualizar**. Los nombres de las salas (como "General", "Juegos", "Música") se cargan como ítems del `ComboBox`.
        - **Evento `SelectedIndexChanged`:** Este evento se dispara cada vez que el usuario **selecciona una sala diferente** del `ComboBox`. Su lógica es crucial para limpiar el `lstChat` y luego cargar los mensajes específicos de la sala seleccionada desde el diccionario `salas`.
        - **Propiedad `SelectedItem`:** Obtiene el **nombre de la sala actualmente elegida** por el usuario. Este valor de cadena se usa como clave para acceder a la lista de mensajes correspondiente en el diccionario `salas`.
    - **`TextBox`**:

        - Name: **txtMensaje**
        - **Utilización:** Campo donde el usuario **escribe el mensaje que desea enviar** en la sala de chat actualmente seleccionada.
        - **Método Clear():** Después de que un mensaje es enviado, **vacía el TextBox** para que el usuario pueda escribir otro.
    - **Button**:
        - Name: **btnEnviar**, Text: "Enviar"
        - **Utilización:** Al hacer clic en este botón, el mensaje que está en txtMensaje se añade al historial interno de la sala seleccionada (en el diccionario salas) y también se muestra en el lstChat visible.
    - **ListBox**:
        - Name: **lstChat**
        - **Utilización:** Es el área principal donde se **muestran los mensajes de la sala de chat activa**. Su contenido cambia dinámicamente según la sala seleccionada en el cmbSalas.
        - **Método Items.Clear():** Se utiliza para **eliminar todos los mensajes actuales del ListBox** cada vez que el usuario cambia de sala, antes de cargar los mensajes de la nueva sala.
        - **Método Items.AddRange(collection):** Carga todos los mensajes de la lista de la sala seleccionada (obtenida del diccionario salas) en el ListBox para su visualización.
        - **Propiedad Items.Add(item):** Añade el mensaje recién enviado a la lista visible de mensajes.
    - **salas (variable de clase Dictionary(Of String, List(Of String))):**
        - **Utilización:** Esta es una estructura de datos clave en el programa. Es un diccionario donde **cada clave (String) es el nombre de una sala, y su valor es una List(Of String) que almacena todos los mensajes de esa sala**. Esto permite mantener historiales de conversación separados para cada sala.

Programa 176. Cliente de chat con videollamadas

- **En qué consiste el ejercicio:** Este programa representa una **implementación conceptual** de cómo se iniciarían las videollamadas en una aplicación de chat. El código proporcionado no implementa la funcionalidad real de videollamada, sino que se centra en **mostrar un mensaje informativo que señala la complejidad** de dicha característica y la necesidad de usar librerías avanzadas externas como AForge o WebRTC para una implementación completa.
- **Controles y propiedades:**
    - **Button**:
        - Name: **btnIniciarVideo**, Text: "Iniciar Videollamada"

        - **Utilización:** Es el botón que el usuario presiona para **"iniciar" una videollamada**. En este ejemplo, su única acción es desencadenar la visualización del `MessageBox` conceptual.
    - **`PictureBox`**:
        - `Name`: **`picVideo`**
        - **Utilización:** Este control se reservaría en una implementación real para **mostrar el flujo de video** capturado desde la cámara web local o recibido de otro participante en la videollamada. En este programa conceptual, no tiene una función activa.
    - **`AxWindowsMediaPlayer`**:
        - `Name`: **`wmpVideo`**
        - **Utilización:** Podría ser utilizado en una implementación real para la **reproducción de video recibido** en una videollamada, o para la gestión de flujos multimedia. En este ejemplo, al igual que `picVideo`, no está activo.
    - **`MessageBox`**:
        - **Utilización:** El elemento central de este programa conceptual. Se utiliza para **mostrar un mensaje al usuario explicando que la funcionalidad de videollamada es compleja** y requeriría el uso de librerías avanzadas como AForge o WebRTC, que están fuera del alcance de este ejemplo básico.

Programa 177. Cliente de chat con llamadas de voz

- **En qué consiste el ejercicio:** Al igual que el programa de videollamadas, este es un **ejemplo conceptual** que ilustra cómo se podrían iniciar y detener las llamadas de voz en una aplicación de chat. El código no proporciona una funcionalidad real de llamada de voz, sino que utiliza `MessageBox` para **indicar de manera abstracta el inicio y fin de una llamada**, y para señalar la necesidad de librerías avanzadas externas para una implementación práctica.
- **Controles y propiedades:**
    - **`Button`**:
        - `Name`: **`btnIniciarVoz`**, `Text`: "Iniciar Llamada"
        - **Utilización:** Es el botón que el usuario presionaría para **"iniciar" una llamada de voz**. Al hacer clic, se muestra un `MessageBox` que indica conceptualmente que la llamada ha comenzado y que la implementación real es más compleja.
        - `Name`: **`btnDetenerVoz`**, `Text`: "Detener"
        - **Utilización:** Es el botón que el usuario presionaría para **"finalizar" una llamada de voz**. Al hacer clic, se muestra un `MessageBox` que indica conceptualmente que la llamada ha sido detenida.
    - **`MessageBox`**:
        - **Utilización:** Se utiliza para **mostrar mensajes conceptuales al usuario**:
            - Cuando se inicia la llamada, informa que la funcionalidad es conceptual y requeriría librerías como NAudio o protocolos VoIP.

- Cuando se detiene la llamada, confirma que la llamada de voz ha sido detenida.

Programa 178. Cliente de chat con notificaciones

- **En qué consiste el ejercicio:** Este programa implementa un sistema de notificaciones que permite **alertar al usuario sobre nuevos mensajes (o eventos) en el chat a través de la bandeja del sistema (system tray)** de Windows. Esto es útil para mantener al usuario informado incluso cuando la aplicación de chat no está en primer plano.
- **Controles y propiedades:**
  - **`NotifyIcon`**:
    - `Name`: **`NotifyIcon1`**, `Icon`: `icono.ico`
    - **Utilización:** Este control permite a la aplicación **mostrar un ícono en la bandeja de notificaciones de Windows** (cerca del reloj). Es un medio discreto para interactuar con el usuario y mostrar notificaciones. La propiedad `Icon` es crucial para especificar el archivo de ícono (`icono.ico`) que se mostrará.
    - **Propiedad `Visible`:** Se establece en `True` (`NotifyIcon1.Visible = True`) para **asegurar que el ícono de la aplicación se muestre** en la bandeja del sistema. Es necesario que el ícono sea visible para que las notificaciones de globo aparezcan.
    - **Método `ShowBalloonTip(timeout, title, text, icon)`:** Este es el método principal para mostrar la notificación. Crea un **mensaje en forma de globo ("balloon tip")** que emerge del ícono en la bandeja.
      - `timeout`: Duración de la notificación en milisegundos (3000 ms = 3 segundos).
      - `title`: El título de la notificación (por ejemplo, "Chat").
      - `text`: El mensaje principal de la notificación (por ejemplo, "Tienes un nuevo mensaje").
      - `icon`: El tipo de ícono que aparece en el globo (`ToolTipIcon.Info` para información general).
  - **`Button`**:
    - `Name`: **`btnNotificar`**, `Text`: "Enviar Notificación"
    - **Utilización:** Al hacer clic en este botón, se **activa la lógica para mostrar la notificación** en la bandeja del sistema, haciendo que aparezca el globo informativo.

Programa 179. Cliente de chat con moderadores

- **En qué consiste el ejercicio:** Este programa introduce una **funcionalidad básica de moderación** para un cliente de chat, permitiendo simular el bloqueo de usuarios. Los nombres de los usuarios bloqueados se añaden a una lista interna. **Es un ejemplo**

conceptual y no incluye la lógica completa para impedir que un usuario bloqueado envíe o reciba mensajes, sino que se centra en el registro del bloqueo.

- **Controles y propiedades:**
    - **`ListBox`**:
        - `Name`: **`lstChat`**
        - **Utilización:** En este contexto, representa el área donde se mostrarían los mensajes de chat. Aunque el código no lo usa para añadir mensajes en este programa específico, se asume que forma parte de la interfaz de chat.
    - **`Button`**:
        - `Name`: **`btnBloquear`**, `Text`: "Bloquear Usuario"
        - **Utilización:** Cuando el moderador hace clic en este botón, el nombre de usuario introducido en `txtUsuario` se añade a la lista de usuarios bloqueados, y se muestra un mensaje de confirmación.
    - **`TextBox`**:
        - `Name`: **`txtUsuario`**
        - **Utilización:** Es el campo donde el moderador **introduce el nombre del usuario** que desea bloquear del chat.
    - **`usuariosBloqueados` (variable de clase `List(Of String)`):**
        - **Utilización:** Una variable declarada a nivel de clase que es una **lista dinámica de cadenas**, diseñada para **almacenar los nombres de los usuarios que han sido bloqueados**. En una implementación completa, esta lista se consultaría para determinar si un usuario tiene permisos para interactuar.
        - **Método `Add(item)`:** Se utiliza para **añadir el nombre del usuario** (obtenido de `txtUsuario.Text`) a la lista de usuarios bloqueados.
    - **`MessageBox`**:
        - **Utilización:** Muestra un mensaje de confirmación al moderador, indicando que el usuario especificado ha sido bloqueado (`"Usuario " & txtUsuario.Text & " bloqueado"`).

Programa 180. Cliente de chat con administración

- **En qué consiste el ejercicio:** Este programa implementa una interfaz básica para la **administración de usuarios en un chat**. Permite agregar nuevos nombres de usuario a una lista visible (`ListBox`) y eliminar usuarios existentes de esa misma lista. **Es una funcionalidad puramente visual**; no interactúa con un sistema de usuarios persistente ni con lógica de chat real, sino que simula la gestión de usuarios.
- **Controles y propiedades:**
    - **`ListBox`**:
        - `Name`: **`lstUsuarios`**

        - **Utilización:** Es el control principal que **muestra la lista actual de usuarios** registrados o gestionados. Los usuarios pueden ser añadidos a esta lista o seleccionados de ella para ser eliminados.
        - **Propiedad `Items.Add(item)`:** Se utiliza para **añadir el nombre de usuario** (obtenido de `txtUsuario.Text`) como un nuevo elemento a la colección de ítems visibles en el `ListBox`.
        - **Propiedad `SelectedItem`:** Cuando el usuario intenta eliminar un usuario, esta propiedad **obtiene el elemento (el nombre de usuario) que está actualmente seleccionado** en el `ListBox`. Esto permite saber qué usuario se quiere eliminar.
        - **Método `Items.Remove(item)`:** Se utiliza para **eliminar un elemento específico** (el `SelectedItem`) de la colección de ítems visibles en el `ListBox`, actualizando la lista de usuarios.
    - **`Button`**:
        - `Name`: **`btnEliminar`**, `Text`: "Eliminar Usuario"
        - **Utilización:** Al hacer clic en este botón, el programa **verifica si hay un usuario seleccionado en `lstUsuarios` y, si lo hay, lo elimina** de la lista.
        - `Name`: **`btnAgregar`**, `Text`: "Agregar Usuario"
        - **Utilización:** Al hacer clic en este botón, el texto introducido en `txtUsuario` se **añade como un nuevo elemento a la lista `lstUsuarios`**, simulando el registro de un nuevo usuario.
    - **`TextBox`**:
        - `Name`: **`txtUsuario`**
        - **Utilización:** Es el campo de entrada donde el administrador **introduce el nombre del nuevo usuario** que desea agregar a la lista.

Programa 181. Cliente de chat con cifrado de mensajes

- **En qué consiste el ejercicio:** Este programa introduce el concepto de **seguridad básica en la comunicación** mediante el cifrado de mensajes en un cliente de chat. Utiliza la codificación **Base64** para transformar el texto plano ingresado por el usuario en una cadena cifrada antes de mostrarla en el historial del chat. Esto simula cómo los mensajes podrían ser transmitidos de forma menos legible para un observador no autorizado.
- **Controles necesarios y su utilización:**
    - **TextBox:** `Name: txtMensaje`
        - **Descripción:** Un control TextBox es un campo de entrada de texto donde el usuario puede escribir información.
        - **Propiedades y Utilización:**
            - **`Name` (txtMensaje):** Es el identificador único del control en el código. Se utiliza para referirse a este TextBox en el código VB.NET.
            - **`Text`:** Almacena el contenido de texto que el usuario ha escrito o que se le ha asignado programáticamente. En este programa,

`txtMensaje.Text` se utiliza para obtener el mensaje que el usuario desea enviar. Después de enviar, se limpia con `txtMensaje.Clear()` para preparar la siguiente entrada.

- **Uso en el programa:** Permite al usuario **escribir el mensaje** que quiere enviar y cifrar.

- **Button:** `Name: btnEnviar`, `Text: "Enviar"`
  - **Descripción:** Un control Button es un elemento interactivo que ejecuta una acción cuando se hace clic en él.
  - **Propiedades y Utilización:**
    - **`Name` (btnEnviar):** Identificador único del botón.
    - **`Text` ("Enviar"):** Es el texto que aparece visiblemente en el botón, indicando su función.
    - **Evento `Click`:** Es el evento principal que se activa cuando el usuario hace clic en el botón.
  - **Uso en el programa:** Al hacer clic en este botón, se activa el evento `btnEnviar_Click`, que se encarga de **recopilar el mensaje, cifrarlo y añadirlo al historial del chat**.
- **ListBox:** `Name: lstChat`
  - **Descripción:** Un control ListBox muestra una lista de elementos entre los cuales el usuario puede seleccionar uno o varios.
  - **Propiedades y Utilización:**
    - **`Name` (lstChat):** Identificador único del control.
    - **`Items`:** Es una colección que contiene todos los elementos que se muestran en el ListBox.
      - **`lstChat.Items.Add(elemento)`:** Se utiliza para **añadir un nuevo elemento** a la lista.
  - **Uso en el programa:** Funciona como el **historial de mensajes del chat**, mostrando cada mensaje (cifrado) a medida que se "envía".

Programa 182. Cliente de chat con recuperación de historial

- **En qué consiste el ejercicio:** Este programa mejora la funcionalidad del chat al permitir que el historial de mensajes se **cargue automáticamente desde un archivo** al inicio de la aplicación y se **guarde de forma persistente** cada vez que se envía un mensaje. Además, ofrece un botón para **recuperar manualmente** el historial guardado en cualquier momento.
- **Controles necesarios y su utilización:**
  - **TextBox:** `Name: txtMensaje`
    - **Descripción:** Campo de entrada de texto.
    - **Propiedades y Utilización:**
      - **`Name` (txtMensaje):** Identificador único.
      - **`Text`:** Contiene el mensaje que el usuario escribe. Se limpia con `txtMensaje.Clear()` después de cada envío.

        - **Uso en el programa:** Para que el usuario **escriba los mensajes** que desea enviar y añadir al historial.
    - **Button:** `Name: btnEnviar`, `Text: "Enviar"`
        - **Descripción:** Botón para ejecutar una acción.
        - **Propiedades y Utilización:**
            - **`Name` (btnEnviar):** Identificador único.
            - **`Text` ("Enviar"):** Texto visible.
            - **Evento `Click`:** Se activa al hacer clic.
        - **Uso en el programa:** Al hacer clic, **añade el mensaje al `ListBox` y lo guarda en el archivo** de historial.
    - **Button:** `Name: btnRecuperar`, `Text: "Recuperar Historial"`
        - **Descripción:** Botón para ejecutar una acción.
        - **Propiedades y Utilización:**
            - **`Name` (btnRecuperar):** Identificador único.
            - **`Text` ("Recuperar Historial"):** Texto visible.
            - **Evento `Click`:** Se activa al hacer clic.
        - **Uso en el programa:** Permite al usuario **recargar el contenido del archivo de historial** en el `ListBox` del chat.
    - **ListBox:** `Name: lstChat`
        - **Descripción:** Muestra una lista de elementos.
        - **Propiedades y Utilización:**
            - **`Name` (lstChat):** Identificador único.
            - **`Items`:** Colección de elementos mostrados.
                - **`lstChat.Items.Add(elemento)`:** Añade un mensaje al historial visible.
                - **`lstChat.Items.Clear()`:** Vacía todos los elementos de la lista, útil antes de cargar un nuevo historial.
                - **`lstChat.Items.AddRange(arrayDeCadenas)`:** Añade una colección de elementos (como las líneas leídas de un archivo) a la lista.
        - **Uso en el programa:** Sirve como la **ventana principal donde se visualizan los mensajes** y el historial, tanto los enviados recientemente como los recuperados del archivo.

Programa 183. Cliente de chat con estadísticas

- **En qué consiste el ejercicio:** Este programa añade una funcionalidad sencilla para **mostrar estadísticas básicas del chat**, específicamente el **número total de mensajes** que se han registrado en el historial visible del `ListBox` en un momento dado.
- **Controles necesarios y su utilización:**
    - **Button:** `Name: btnEstadisticas`, `Text: "Mostrar Estadísticas"`

        - **Descripción:** Botón para ejecutar una acción.
        - **Propiedades y Utilización:**
            - **`Name` (btnEstadisticas):** Identificador único.
            - **`Text` ("Mostrar Estadísticas"):** Texto visible.
            - **Evento `Click`:** Se activa al hacer clic.
        - **Uso en el programa:** Al hacer clic, **calcula y muestra el total de mensajes** presentes en el `lstChat`.
    - **ListBox:** `Name: lstChat`
        - **Descripción:** Muestra una lista de elementos.
        - **Propiedades y Utilización:**
            - **`Name` (lstChat):** Identificador único.
            - **`Items.Count`:** Es una propiedad que devuelve el **número total de elementos** (mensajes en este caso) que hay actualmente en la colección `Items` del ListBox.
        - **Uso en el programa:** Contiene la **lista de mensajes** de chat, que son la base para calcular la estadística.

Programa 184. Cliente de chat con exportación de mensajes

- **En qué consiste el ejercicio:** Este programa permite al usuario **exportar todos los mensajes** actualmente visibles en el `ListBox` del chat a un **archivo de texto plano**. El usuario puede elegir la ubicación y el nombre del archivo de salida mediante un cuadro de diálogo estándar.
- **Controles necesarios y su utilización:**
    - **Button:** `Name: btnExportar`, `Text: "Exportar"`
        - **Descripción:** Botón para ejecutar una acción.
        - **Propiedades y Utilización:**
            - **`Name` (btnExportar):** Identificador único.
            - **`Text` ("Exportar"):** Texto visible.
            - **Evento `Click`:** Se activa al hacer clic.
        - **Uso en el programa:** Inicia el proceso de **exportar los mensajes** a un archivo de texto.
    - **SaveFileDialog:** `Name: SaveFileDialog1`
        - **Descripción:** Un control no visual que abre un cuadro de diálogo estándar de Windows para que el usuario seleccione una ubicación y un nombre de archivo para guardar datos.
        - **Propiedades y Utilización:**
            - **`Name` (SaveFileDialog1):** Identificador único.
            - **`ShowDialog()`:** Abre el cuadro de diálogo. Devuelve `DialogResult.OK` si el usuario selecciona un archivo y hace clic en "Guardar".

        - **`FileName`:** Después de que `ShowDialog()` devuelve `DialogResult.OK`, esta propiedad contiene la ruta completa (directorio y nombre) del archivo que el usuario ha especificado.
    - **Uso en el programa:** Permite al usuario **elegir dónde y con qué nombre guardar el archivo** que contendrá los mensajes exportados.
- **ListBox:** `Name: lstChat`
    - **Descripción:** Muestra una lista de elementos.
    - **Propiedades y Utilización:**
        - **`Name` (lstChat):** Identificador único.
        - **`Items`:** Colección de todos los elementos en la lista.
            - **`lstChat.Items.Cast(Of String)()`:** Este método (requiere `Imports System.Linq`, aunque no se muestra explícitamente en la fuente para este programa pero es una práctica común para manipular colecciones) convierte la colección `Items` (que es de tipo `Object`) en una colección de tipo `String`, lo cual es necesario para métodos de archivo como `File.WriteAllLines()` que esperan una colección de cadenas.
    - **Uso en el programa:** Contiene la **lista de mensajes** de chat que se van a exportar. El contenido de cada elemento de esta lista será una línea en el archivo de texto de salida.

Programa 185. Cliente de chat con importación de mensajes

- **En qué consiste el ejercicio:** Este programa complementa la funcionalidad de exportación al permitir que el usuario **importe mensajes de chat desde un archivo de texto** previamente guardado. El contenido del archivo se carga y se muestra en el `ListBox` del chat.
- **Controles necesarios y su utilización:**
    - **Button:** `Name: btnImportar`, `Text: "Importar"`
        - **Descripción:** Botón para ejecutar una acción.
        - **Propiedades y Utilización:**
            - **`Name` (btnImportar):** Identificador único.
            - **`Text` ("Importar"):** Texto visible.
            - **Evento `Click`:** Se activa al hacer clic.
        - **Uso en el programa:** Inicia el proceso de **importar mensajes** desde un archivo de texto.
    - **OpenFileDialog:** `Name: OpenFileDialog1`
        - **Descripción:** Un control no visual que abre un cuadro de diálogo estándar de Windows para que el usuario seleccione un archivo existente.
        - **Propiedades y Utilización:**
            - **`Name` (OpenFileDialog1):** Identificador único.

        - **`ShowDialog()`:** Abre el cuadro de diálogo. Devuelve `DialogResult.OK` si el usuario selecciona un archivo y hace clic en "Abrir".
        - **`FileName`:** Después de que `ShowDialog()` devuelve `DialogResult.OK`, esta propiedad contiene la ruta completa del archivo que el usuario ha seleccionado.
    - **Uso en el programa:** Permite al usuario **elegir el archivo de texto** desde el cual se importarán los mensajes.
- **ListBox:** `Name: lstChat`
    - **Descripción:** Muestra una lista de elementos.
    - **Propiedades y Utilización:**
        - **`Name` (lstChat):** Identificador único.
        - **`Items`:** Colección de elementos mostrados.
            - **`lstChat.Items.Clear()`:** Limpia todos los elementos actuales del ListBox antes de cargar los nuevos, asegurando que solo se muestren los mensajes importados.
            - **`lstChat.Items.AddRange(arrayDeCadenas)`:** Añade una colección completa de elementos (en este caso, todas las líneas leídas del archivo) a la lista de una sola vez.
    - **Uso en el programa:** Sirve como la **ventana principal donde se visualizarán los mensajes importados**, reemplazando cualquier historial anterior.

Programa 186. Cliente de correo electrónico básico

- **En qué consiste el ejercicio:** Este programa representa una **implementación conceptual de un cliente de correo electrónico muy básico**. Permite al usuario redactar un correo (especificando destinatario, asunto y cuerpo), pero **no realiza el envío real**. En su lugar, informa al usuario que el correo ha sido "preparado" y que el envío real requeriría una configuración SMTP.
- **Controles necesarios y su utilización:**
    - **TextBox:** `Name: txtPara`
        - **Descripción:** Campo de entrada de texto.
        - **Propiedades y Utilización:**
            - **`Name` (txtPara):** Identificador único.
            - **`Text`:** Contiene la dirección de correo electrónico del destinatario ingresada por el usuario.
        - **Uso en el programa:** Para que el usuario **introduzca la dirección de correo electrónico del destinatario**.
    - **TextBox:** `Name: txtAsunto`
        - **Descripción:** Campo de entrada de texto.
        - **Propiedades y Utilización:**

        - **`Name` (txtAsunto):** Identificador único.
        - **`Text`:** Contiene el asunto del correo ingresado por el usuario.
    - **Uso en el programa:** Para que el usuario **escriba el asunto del correo electrónico**.
- **TextBox:** `Name: txtCuerpo`, `Multiline: True`
    - **Descripción:** Un TextBox configurado para aceptar múltiples líneas de texto.
    - **Propiedades y Utilización:**
        - **`Name` (txtCuerpo):** Identificador único.
        - **`Multiline` (True):** Esta propiedad, cuando se establece en `True`, permite que el TextBox muestre y acepte texto en varias líneas, lo cual es esencial para el cuerpo de un correo.
        - **`Text`:** Contiene el cuerpo del mensaje ingresado por el usuario.
    - **Uso en el programa:** Para que el usuario **redacte el cuerpo del mensaje de correo electrónico**.
- **Button:** `Name: btnEnviar`, `Text: "Enviar"`
    - **Descripción:** Botón para ejecutar una acción.
    - **Propiedades y Utilización:**
        - **`Name` (btnEnviar):** Identificador único.
        - **`Text` ("Enviar"):** Texto visible.
        - **Evento `Click`:** Se activa al hacer clic.
    - **Uso en el programa:** Al hacer clic, **simula la preparación del correo** recopilando la información de los TextBox en un objeto `MailMessage`.

Programa 187. Cliente de correo con envío SMTP

- **En qué consiste el ejercicio:** Este programa va un paso más allá del básico y **implementa el envío real de correos electrónicos** utilizando el protocolo SMTP (Simple Mail Transfer Protocol). Requiere que el usuario proporcione el servidor SMTP, su correo electrónico (como remitente) y su contraseña para la autenticación, y luego envía el mensaje al destinatario especificado.
- **Controles necesarios y su utilización:**
    - **TextBox:** `Name: txtPara`
        - **Descripción:** Campo de entrada de texto.
        - **Propiedades y Utilización:**
            - **`Name` (txtPara):** Identificador único.
            - **`Text`:** Contiene la dirección de correo electrónico del destinatario.
        - **Uso en el programa:** Para que el usuario **introduzca la dirección de correo del destinatario**.
    - **TextBox:** `Name: txtAsunto`
        - **Descripción:** Campo de entrada de texto.
        - **Propiedades y Utilización:**

        - **`Name` (txtAsunto):** Identificador único.
        - **`Text`:** Contiene el asunto del correo.
    - **Uso en el programa:** Para que el usuario **escriba el asunto del correo electrónico**.
- **TextBox:** `Name: txtCuerpo`, `Multiline: True`
    - **Descripción:** TextBox configurado para múltiples líneas.
    - **Propiedades y Utilización:**
        - **`Name` (txtCuerpo):** Identificador único.
        - **`Multiline` (True):** Permite texto de varias líneas.
        - **`Text`:** Contiene el cuerpo del mensaje.
    - **Uso en el programa:** Para que el usuario **redacte el cuerpo del mensaje**.
- **TextBox:** `Name: txtSMTP` (servidor SMTP)
    - **Descripción:** Campo de entrada de texto.
    - **Propiedades y Utilización:**
        - **`Name` (txtSMTP):** Identificador único.
        - **`Text`:** Contiene la dirección del servidor SMTP (por ejemplo, `smtp.gmail.com` para Gmail).
    - **Uso en el programa:** Para que el usuario **especifique el servidor SMTP** a través del cual se enviará el correo.
- **TextBox:** `Name: txtUsuario` (correo remitente)
    - **Descripción:** Campo de entrada de texto.
    - **Propiedades y Utilización:**
        - **`Name` (txtUsuario):** Identificador único.
        - **`Text`:** Contiene la dirección de correo electrónico del remitente.
    - **Uso en el programa:** Para que el usuario **introduzca su propia dirección de correo electrónico**, que se utilizará como remitente y para la autenticación.
- **TextBox:** `Name: txtPassword`, `PasswordChar: *`
    - **Descripción:** Campo de entrada de texto diseñado para contraseñas.
    - **Propiedades y Utilización:**
        - **`Name` (txtPassword):** Identificador único.
        - **`PasswordChar` (`"*"`):** Cuando se establece, cualquier carácter que el usuario escriba en el TextBox se muestra como el carácter especificado (en este caso, un asterisco `*`), ocultando la contraseña para mayor seguridad.
        - **`Text`:** Contiene la contraseña ingresada, aunque oculta.
    - **Uso en el programa:** Para que el usuario **ingrese su contraseña de correo electrónico** de forma segura, necesaria para autenticarse con el servidor SMTP.
- **Button:** `Name: btnEnviar`, `Text: "Enviar"`
    - **Descripción:** Botón para ejecutar una acción.
    - **Propiedades y Utilización:**
        - **`Name` (btnEnviar):** Identificador único.

        - **`Text` ("Enviar"):** Texto visible.
        - **Evento `Click`:** Se activa al hacer clic.
    - **Uso en el programa:** Al hacer clic, **inicia el proceso de conexión al servidor SMTP, autenticación y envío del correo electrónico**.

Programa 188. Cliente de correo con recepción POP3

- **En qué consiste el ejercicio:** Este programa es una **implementación conceptual para la recepción de correos electrónicos** utilizando el protocolo POP3 (Post Office Protocol 3). Se asume el uso de una librería externa como OpenPop.NET (que debería ser instalada vía NuGet). Permite al usuario conectarse a un servidor POP3, autenticarse y listar los asuntos de los correos disponibles en su bandeja de entrada.
- **Controles necesarios y su utilización:**
    - **TextBox:** `Name: txtServidor`
        - **Descripción:** Campo de entrada de texto.
        - **Propiedades y Utilización:**
            - **`Name` (txtServidor):** Identificador único.
            - **`Text`:** Contiene la dirección del servidor POP3 (por ejemplo, `pop.gmail.com`).
        - **Uso en el programa:** Para que el usuario **especifique la dirección del servidor POP3** al que desea conectarse para recibir correos.
    - **TextBox:** `Name: txtUsuario`
        - **Descripción:** Campo de entrada de texto.
        - **Propiedades y Utilización:**
            - **`Name` (txtUsuario):** Identificador único.
            - **`Text`:** Contiene el nombre de usuario (dirección de correo) para la autenticación.
        - **Uso en el programa:** Para que el usuario **ingrese su nombre de usuario o dirección de correo electrónico** para acceder a su buzón.
    - **TextBox:** `Name: txtPassword`, `PasswordChar: *`
        - **Descripción:** Campo de entrada de texto diseñado para contraseñas.
        - **Propiedades y Utilización:**
            - **`Name` (txtPassword):** Identificador único.
            - **`PasswordChar` ("*"):** Oculta los caracteres escritos, mostrando asteriscos.
            - **`Text`:** Contiene la contraseña ingresada, aunque oculta.
        - **Uso en el programa:** Para que el usuario **introduzca su contraseña de forma segura** para autenticarse con el servidor POP3.
    - **Button:** `Name: btnRecibir`, `Text: "Recibir"`
        - **Descripción:** Botón para ejecutar una acción.
        - **Propiedades y Utilización:**

        - **`Name` (btnRecibir):** Identificador único.
        - **`Text` ("Recibir"):** Texto visible.
        - **Evento `Click`:** Se activa al hacer clic.
    - **Uso en el programa:** Al hacer clic, **inicia el proceso de conexión, autenticación y descarga de los asuntos** de los correos disponibles en el servidor POP3, mostrándolos en el `ListBox`.
- **ListBox:** `Name: lstCorreos`
    - **Descripción:** Muestra una lista de elementos.
    - **Propiedades y Utilización:**
        - **`Name` (lstCorreos):** Identificador único.
        - **`Items`:** Colección de elementos mostrados.
            - **`lstCorreos.Items.Clear()`:** Limpia la lista antes de cargar nuevos correos.
            - **`lstCorreos.Items.Add(elemento)`:** Añade el asunto de cada correo como un nuevo elemento a la lista.
    - **Uso en el programa:** Sirve para **mostrar los asuntos de los correos electrónicos** recuperados del servidor POP3.

Programa 189. Cliente de correo con conexión IMAP

- **En qué consiste el ejercicio:** Este programa es una **implementación conceptual para interactuar con un servidor de correo IMAP** (Internet Message Access Protocol). Al igual que el programa POP3, asume el uso de una librería externa (MailKit, instalada vía NuGet). Permite al usuario conectarse a un servidor IMAP, autenticarse, abrir la bandeja de entrada y **listar los asuntos de los correos**.
- **Controles necesarios y su utilización:**
    - **TextBox:** `Name: txtServidor`
        - **Descripción:** Campo de entrada de texto.
        - **Propiedades y Utilización:**
            - **`Name` (txtServidor):** Identificador único.
            - **`Text`:** Contiene la dirección del servidor IMAP (por ejemplo, `imap.gmail.com`).
        - **Uso en el programa:** Para que el usuario **especifique la dirección del servidor IMAP** al que desea conectarse.
    - **TextBox:** `Name: txtUsuario`
        - **Descripción:** Campo de entrada de texto.
        - **Propiedades y Utilización:**
            - **`Name` (txtUsuario):** Identificador único.
            - **`Text`:** Contiene el nombre de usuario (dirección de correo) para la autenticación.
        - **Uso en el programa:** Para que el usuario **ingrese su nombre de usuario o dirección de correo electrónico** para acceder a su buzón IMAP.

- **TextBox:** `Name: txtPassword`, `PasswordChar: *`
    - **Descripción:** Campo de entrada de texto diseñado para contraseñas.
    - **Propiedades y Utilización:**
        - **`Name` (txtPassword):** Identificador único.
        - **`PasswordChar` ("*"):** Oculta los caracteres escritos.
        - **`Text`:** Contiene la contraseña ingresada, aunque oculta.
    - **Uso en el programa:** Para que el usuario **introduzca su contraseña de forma segura** para autenticarse con el servidor IMAP.
- **Button:** `Name: btnRecibir`, `Text: "Recibir"`
    - **Descripción:** Botón para ejecutar una acción.
    - **Propiedades y Utilización:**
        - **`Name` (btnRecibir):** Identificador único.
        - **`Text` ("Recibir"):** Texto visible.
        - **Evento `Click`:** Se activa al hacer clic.
    - **Uso en el programa:** Al hacer clic, **inicia el proceso de conexión, autenticación y descarga de los asuntos** de los correos de la bandeja de entrada del servidor IMAP, mostrándolos en el `ListBox`.
- **ListBox:** `Name: lstCorreos`
    - **Descripción:** Muestra una lista de elementos.
    - **Propiedades y Utilización:**
        - **`Name` (lstCorreos):** Identificador único.
        - **`Items`:** Colección de elementos mostrados.
            - **`lstCorreos.Items.Clear()`:** Limpia la lista antes de cargar nuevos correos.
            - **`lstCorreos.Items.Add(elemento)`:** Añade el asunto de cada correo como un nuevo elemento a la lista.
    - **Uso en el programa:** Sirve para **mostrar los asuntos de los correos electrónicos** recuperados de la bandeja de entrada IMAP.

Programa 190. Cliente de correo con adjuntos

- **En qué consiste el ejercicio:** Este programa amplía la funcionalidad del cliente de correo para permitir al usuario **adjuntar uno o más archivos** a un correo electrónico antes de su envío conceptual. Muestra los archivos seleccionados en una lista y los incluye al "preparar" el mensaje.
- **Controles necesarios y su utilización:**
    - **TextBox:** `Name: txtPara`
        - **Descripción:** Campo de entrada de texto.
        - **Propiedades y Utilización:**
            - **`Name` (txtPara):** Identificador único.
            - **`Text`:** Contiene la dirección del destinatario.

    - **Uso en el programa:** Para que el usuario **introduzca la dirección de correo del destinatario**.
- **TextBox:** `Name: txtAsunto`
    - **Descripción:** Campo de entrada de texto.
    - **Propiedades y Utilización:**
        - **`Name` (txtAsunto):** Identificador único.
        - **`Text`:** Contiene el asunto del correo.
    - **Uso en el programa:** Para que el usuario **escriba el asunto del correo electrónico**.
- **TextBox:** `Name: txtCuerpo`, `Multiline: True`
    - **Descripción:** TextBox para texto de múltiples líneas.
    - **Propiedades y Utilización:**
        - **`Name` (txtCuerpo):** Identificador único.
        - **`Multiline` (True):** Permite el texto en varias líneas.
        - **`Text`:** Contiene el cuerpo del mensaje.
    - **Uso en el programa:** Para que el usuario **redacte el cuerpo del mensaje**.
- **Button:** `Name: btnAdjuntar`, `Text: "Adjuntar Archivo"`
    - **Descripción:** Botón para ejecutar una acción.
    - **Propiedades y Utilización:**
        - **`Name` (btnAdjuntar):** Identificador único.
        - **`Text` ("Adjuntar Archivo"):** Texto visible.
        - **Evento `Click`:** Se activa al hacer clic.
    - **Uso en el programa:** Abre un cuadro de diálogo para que el usuario **seleccione archivos locales para adjuntar** al correo.
- **Button:** `Name: btnEnviar`, `Text: "Enviar"`
    - **Descripción:** Botón para ejecutar una acción.
    - **Propiedades y Utilización:**
        - **`Name` (btnEnviar):** Identificador único.
        - **`Text` ("Enviar"):** Texto visible.
        - **Evento `Click`:** Se activa al hacer clic.
    - **Uso en el programa:** Recopila toda la información del correo (destinatario, asunto, cuerpo y archivos adjuntos) y **simula su preparación para el envío**.
- **OpenFileDialog:** `Name: OpenFileDialog1`
    - **Descripción:** Control no visual para seleccionar archivos.
    - **Propiedades y Utilización:**
        - **`Name` (OpenFileDialog1):** Identificador único.
        - **`ShowDialog()`:** Muestra el cuadro de diálogo. Devuelve `DialogResult.OK` si se selecciona un archivo.
        - **`FileName`:** Ruta completa del archivo seleccionado.
    - **Uso en el programa:** Se utiliza para permitir al usuario **navegar y seleccionar los archivos** que desea adjuntar al correo electrónico.
- **ListBox:** `Name: lstAdjuntos`

- **Descripción:** Muestra una lista de elementos.
- **Propiedades y Utilización:**
  - **`Name` (lstAdjuntos):** Identificador único.
  - **`Items`:** Colección de elementos mostrados.
    - **`lstAdjuntos.Items.Add(elemento)`:** Añade el nombre del archivo adjunto seleccionado a esta lista, para que el usuario pueda ver qué archivos ha elegido.
- **Uso en el programa:** Muestra una **lista visible de los nombres de los archivos que han sido seleccionados** para ser adjuntados al correo.

Programa 191. Juego de preguntas con ranking

- **En qué consiste el ejercicio:** Este programa implementa un **juego de preguntas y respuestas** que lleva un registro del **puntaje actual del jugador** y lo muestra en un `ListBox` como un **ranking**. Cada vez que el jugador responde a una pregunta y avanza, su puntaje actual se registra, permitiendo visualizar la progresión o diferentes intentos.
- **Controles necesarios y su utilización:**
  - **Label:** `Name: lblPregunta`
    - **Descripción:** Un control Label se utiliza para mostrar texto estático o dinámico al usuario.
    - **Propiedades y Utilización:**
      - **`Name` (lblPregunta):** Identificador único del control.
      - **`Text`:** Almacena la cadena de texto que se muestra. En este programa, `lblPregunta.Text` se utiliza para **mostrar la pregunta actual** del juego.
    - **Uso en el programa:** Presenta las preguntas al jugador.
  - **TextBox:** `Name: txtRespuesta`
    - **Descripción:** Un campo de entrada de texto donde el usuario puede escribir información.
    - **Propiedades y Utilización:**
      - **`Name` (txtRespuesta):** Identificador único.
      - **`Text`:** Almacena el contenido escrito por el usuario. `txtRespuesta.Text` se utiliza para **capturar la respuesta del jugador**.
    - **Uso en el programa:** Permite al jugador introducir su respuesta a la pregunta.
  - **Button:** `Name: btnVerificar`, `Text: "Verificar"`
    - **Descripción:** Un control Button ejecuta una acción cuando se hace clic en él.
    - **Propiedades y Utilización:**
      - **`Name` (btnVerificar):** Identificador único.
      - **`Text` ("Verificar"):** Texto visible en el botón.

        - **Evento `Click`:** Se activa al hacer clic. En este evento, el programa **comprueba la respuesta**, actualiza el `puntajeActual`, lo añade al `lstRanking`, y avanza a la siguiente pregunta si la hay.
    - **Uso en el programa:** Inicia el proceso de evaluación de la respuesta y actualización del juego.
- **ListBox:** `Name: lstRanking`
    - **Descripción:** Muestra una lista de elementos que el usuario puede ver o seleccionar.
    - **Propiedades y Utilización:**
        - **`Name` (lstRanking):** Identificador único.
        - **`Items`:** Colección de elementos que se muestran en el ListBox.
            - **`lstRanking.Items.Clear()`:** Elimina todos los elementos de la lista, útil para refrescar el ranking.
            - **`lstRanking.Items.Add(elemento)`:** Añade un nuevo elemento (en este caso, un puntaje) a la lista.
    - **Uso en el programa: Muestra el ranking de puntajes** del jugador a lo largo del juego.

Programa 192. Juego de preguntas con exportación de resultados

- **En qué consiste el ejercicio:** Este programa permite al usuario **exportar los resultados de un juego de preguntas** (elementos presentes en un `ListBox`) a un **archivo de texto** en el sistema. Esto es útil para guardar un registro de las partidas o de los puntajes.
- **Controles necesarios y su utilización:**
    - **Button:** `Name: btnExportar`, `Text: "Exportar Resultados"`
        - **Descripción:** Botón para ejecutar una acción.
        - **Propiedades y Utilización:**
            - **`Name` (btnExportar):** Identificador único.
            - **`Text` ("Exportar Resultados"):** Texto visible en el botón.
            - **Evento `Click`:** Se activa al hacer clic. Este evento **inicia el proceso de guardar el contenido** del `ListBox` en un archivo de texto.
        - **Uso en el programa:** Sirve para iniciar la acción de exportación de los datos.
    - **ListBox:** `Name: lstResultados`
        - **Descripción:** Muestra una lista de elementos.
        - **Propiedades y Utilización:**
            - **`Name` (lstResultados):** Identificador único.
            - **`Items`:** Colección de elementos mostrados.
                - **`lstResultados.Items.Cast(Of String)()`:** Este método (que puede requerir `Imports System.Linq` para una conversión adecuada de `Object` a `String`) se utiliza para **convertir la colección de elementos** del `ListBox` a un

formato `IEnumerable(Of String)` que la función `File.WriteAllLines()` puede procesar directamente.
    - **Uso en el programa:** Contiene la **lista de resultados** del juego de preguntas que se va a exportar. Cada elemento en esta lista se escribirá como una línea separada en el archivo de texto.

Programa 193. Juego de preguntas con importación de resultados

- **En qué consiste el ejercicio:** Este programa complementa la funcionalidad de exportación, permitiendo al usuario **importar resultados de un juego de preguntas** desde un **archivo de texto** previamente guardado. Los resultados importados se cargan y se muestran en un `ListBox`.
- **Controles necesarios y su utilización:**
    - **Button:** `Name: btnImportar`, `Text: "Importar Resultados"`
        - **Descripción:** Botón para ejecutar una acción.
        - **Propiedades y Utilización:**
            - **`Name` (btnImportar):** Identificador único.
            - **`Text` ("Importar Resultados"):** Texto visible.
            - **Evento `Click`:** Se activa al hacer clic. Este evento **desencadena la apertura del cuadro de diálogo para seleccionar el archivo** y, posteriormente, la carga de su contenido.
        - **Uso en el programa:** Inicia la acción de importación de los resultados.
    - **OpenFileDialog:** `Name: OpenFileDialog1`
        - **Descripción:** Un control no visual que presenta un cuadro de diálogo estándar de Windows para que el usuario seleccione uno o varios archivos existentes.
        - **Propiedades y Utilización:**
            - **`Name` (OpenFileDialog1):** Identificador único.
            - **`ShowDialog()`:** Abre el cuadro de diálogo de selección de archivo. Devuelve `DialogResult.OK` si el usuario selecciona un archivo y hace clic en "Abrir".
            - **`FileName`:** Después de que `ShowDialog()` devuelve `DialogResult.OK`, esta propiedad contiene la **ruta completa del archivo seleccionado** por el usuario.
        - **Uso en el programa:** Permite al usuario **elegir el archivo de texto** del que se importarán los resultados.
    - **ListBox:** `Name: lstResultados`
        - **Descripción:** Muestra una lista de elementos.
        - **Propiedades y Utilización:**
            - **`Name` (lstResultados):** Identificador único.
            - **`Items`:** Colección de elementos mostrados.

- **`lstResultados.Items.Clear()`:** Vacía todos los elementos actuales de la lista, asegurando que solo se muestren los resultados importados.
- **`lstResultados.Items.AddRange(arrayDeCadenas)`:** Añade una colección completa de elementos (en este caso, todas las líneas leídas del archivo) a la lista de una sola vez.

- **Uso en el programa:** Sirve como la **ventana principal donde se visualizarán los resultados importados**, reemplazando cualquier contenido anterior.

Programa 194. Juego de preguntas con estadísticas

- **En qué consiste el ejercicio:** Este programa enriquece un juego de preguntas al proporcionar **estadísticas en tiempo real**, mostrando al usuario el **número de respuestas correctas** y el **total de intentos** realizados hasta el momento.
- **Controles necesarios y su utilización:**
  - **Label:** `Name: lblPregunta`
    - **Descripción:** Un control Label muestra texto.
    - **Propiedades y Utilización:**
      - **`Name` (lblPregunta):** Identificador único.
      - **`Text`:** Muestra la pregunta actual o indica el fin del juego.
    - **Uso en el programa:** Presenta las preguntas al jugador.
  - **TextBox:** `Name: txtRespuesta`
    - **Descripción:** Campo de entrada de texto.
    - **Propiedades y Utilización:**
      - **`Name` (txtRespuesta):** Identificador único.
      - **`Text`:** Captura la respuesta del jugador. Se sugiere `txtRespuesta.Clear()` después de cada verificación para una buena práctica.
    - **Uso en el programa:** Permite al jugador introducir su respuesta.
  - **Button:** `Name: btnVerificar`, `Text: "Verificar"`
    - **Descripción:** Botón para ejecutar una acción.
    - **Propiedades y Utilización:**
      - **`Name` (btnVerificar):** Identificador único.
      - **`Text` ("Verificar"):** Texto visible.
      - **Evento `Click`:** Se activa al hacer clic. En este evento, el programa **incrementa el contador de intentos**, evalúa si la respuesta es correcta (incrementando el contador `correctas` si lo es), y avanza a la siguiente pregunta. También **actualiza las estadísticas** mostradas.
    - **Uso en el programa:** Inicia el proceso de verificación de la respuesta y actualización de las estadísticas.
  - **Label:** `Name: lblEstadisticas`

- **Descripción:** Un control Label muestra texto.
- **Propiedades y Utilización:**
  - **`Name` (lblEstadisticas):** Identificador único.
  - **`Text`:** Se actualiza para **mostrar el número de respuestas correctas y el total de intentos**.
- **Uso en el programa:** Muestra al jugador un resumen en tiempo real de su rendimiento en el juego.

Programa 195. Juego de preguntas con gráficos

- **En qué consiste el ejercicio:** Este programa proporciona una **representación visual de los resultados** de un juego de preguntas mediante el uso de un control `Chart`. En este caso, se muestra un **gráfico de columnas** que representa los datos de respuestas correctas e incorrectas.
- **Controles necesarios y su utilización:**
  - **Chart:** `Name: chartResultados`
    - **Descripción:** Un control Chart permite mostrar datos numéricos de forma gráfica (barras, líneas, pasteles, etc.). Requiere la referencia `System.Windows.Forms.DataVisualization`.
    - **Propiedades y Utilización:**
      - **`Name` (chartResultados):** Identificador único.
      - **`Series`:** Es una colección que contiene las diferentes series de datos que se graficarán.
        - **`chartResultados.Series.Clear()`:** Elimina todas las series de datos existentes en el gráfico antes de añadir nuevas, asegurando que el gráfico se refresque.
        - **`chartResultados.Series.Add("NombreSerie")`:** Añade una nueva serie de datos al gráfico, identificada por un nombre.
        - **`chartResultados.Series("NombreSerie").Points.AddXY(xValue, yValue)`:** Añade un punto de datos a una serie específica. `xValue` es la etiqueta en el eje X (categoría) y `yValue` es el valor numérico en el eje Y (magnitud).
      - **`ChartType`:** Aunque no se establece explícitamente en el `chartResultados` en la fuente (se asume configuración en diseño o se establece en la serie), las series individuales pueden tener su propio tipo de gráfico, como `DataVisualization.Charting.SeriesChartType.Column`.
    - **Uso en el programa: Visualiza el conteo de respuestas correctas e incorrectas** en un formato de gráfico de columnas.
  - **Button:** `Name: btnActualizar`, `Text: "Actualizar Gráfico"`

- **Descripción:** Botón para ejecutar una acción.
- **Propiedades y Utilización:**
  - **`Name` (btnActualizar):** Identificador único.
  - **`Text` ("Actualizar Gráfico"):** Texto visible.
  - **Evento `Click`:** Se activa al hacer clic. Este evento **limpia el gráfico existente y lo repuebla con los datos** actuales (correctas e incorrectas).
- **Uso en el programa:** Permite al usuario refrescar la visualización del gráfico con los últimos datos disponibles.

Programa 196. Juego de preguntas con personalización

- **En qué consiste el ejercicio:** Este programa introduce una funcionalidad de **personalización**, permitiendo al usuario **añadir sus propias preguntas** a una lista en el formulario. Esto hace el juego más interactivo y extensible por parte del usuario.
- **Controles necesarios y su utilización:**
  - **TextBox:** `Name: txtPregunta`
    - **Descripción:** Campo de entrada de texto.
    - **Propiedades y Utilización:**
      - **`Name` (txtPregunta):** Identificador único.
      - **`Text`:** Contiene el texto de la pregunta que el usuario desea agregar. Después de añadir la pregunta, se usa `txtPregunta.Clear()` para vaciarlo, preparando la siguiente entrada.
    - **Uso en el programa:** Permite al usuario **escribir el contenido de una nueva pregunta**.
  - **Button:** `Name: btnAgregarPregunta`, `Text: "Agregar Pregunta"`
    - **Descripción:** Botón para ejecutar una acción.
    - **Propiedades y Utilización:**
      - **`Name` (btnAgregarPregunta):** Identificador único.
      - **`Text` ("Agregar Pregunta"):** Texto visible.
      - **Evento `Click`:** Se activa al hacer clic. Este evento **toma el texto del `txtPregunta` y lo añade a la lista de preguntas** (`lstPreguntas`).
    - **Uso en el programa:** Inicia la acción de agregar una pregunta personalizada al juego.
  - **ListBox:** `Name: lstPreguntas`
    - **Descripción:** Muestra una lista de elementos.
    - **Propiedades y Utilización:**
      - **`Name` (lstPreguntas):** Identificador único.
      - **`Items`:** Colección de elementos mostrados.
        - **`lstPreguntas.Items.Add(elemento)`:** Añade el texto de una nueva pregunta a la lista visible.

- **Uso en el programa: Muestra visualmente todas las preguntas** que el usuario ha añadido al juego.

Programa 197. Juego de preguntas con guardado automático

- **En qué consiste el ejercicio:** Este programa mejora la persistencia de datos en un juego de preguntas al **guardar automáticamente las respuestas del usuario** en un archivo de texto en intervalos regulares. Además, ofrece un botón para un **guardado manual**.
- **Controles necesarios y su utilización:**
  - **TextBox:** `Name: txtRespuesta`
    - **Descripción:** Campo de entrada de texto.
    - **Propiedades y Utilización:**
      - **`Name` (txtRespuesta):** Identificador único.
      - **`Text`:** Contiene la respuesta actual del usuario que se va a guardar.
    - **Uso en el programa:** Permite al usuario **introducir sus respuestas**, las cuales serán guardadas.
  - **Button:** `Name: btnGuardar`, `Text: "Guardar"`
    - **Descripción:** Botón para ejecutar una acción.
    - **Propiedades y Utilización:**
      - **`Name` (btnGuardar):** Identificador único.
      - **`Text` ("Guardar"):** Texto visible.
      - **Evento `Click`:** Se activa al hacer clic. Este evento **guarda el contenido de `txtRespuesta` en el archivo** de "respuestas.txt".
    - **Uso en el programa:** Permite al usuario realizar un **guardado manual** de su respuesta.
  - **Timer:** `Name: Timer1`, `Interval: 5000` (milisegundos)
    - **Descripción:** Un control no visual que genera un evento a intervalos regulares.
    - **Propiedades y Utilización:**
      - **`Name` (Timer1):** Identificador único.
      - **`Interval` (5000):** Define el tiempo en milisegundos entre cada disparo del evento `Tick` (5000 ms = 5 segundos).
      - **`Enabled` (configuración inicial, no en la fuente, pero común):** Si está en `True`, el temporizador se inicia automáticamente.
      - **`Start()`:** Inicia el temporizador.
      - **Evento `Tick`:** Se dispara cada vez que transcurre el `Interval`. En este programa, el evento `Timer1_Tick` se encarga del **guardado automático**, añadiendo el texto de `txtRespuesta` al archivo.
    - **Uso en el programa:** Implementa el **guardado automático** de las respuestas del usuario.

Programa 198. Juego de preguntas con recuperación de partida

- **En qué consiste el ejercicio:** Este programa se centra en la **persistencia del progreso del juego**. Permite al usuario **guardar el estado actual de la partida** (específicamente, la pregunta en la que se encuentra) y **recuperarla más tarde**, cargando el juego justo en ese punto.
- **Controles necesarios y su utilización:**
    - **Button:** `Name: btnGuardar`, `Text: "Guardar Partida"`
        - **Descripción:** Botón para ejecutar una acción.
        - **Propiedades y Utilización:**
            - **`Name` (btnGuardar):** Identificador único.
            - **`Text` ("Guardar Partida"):** Texto visible.
            - **Evento `Click`:** Se activa al hacer clic. Este evento **guarda el valor del índice de la pregunta actual** (`indice`) en un archivo de texto llamado "partida.txt".
        - **Uso en el programa:** Permite al usuario **salvar el progreso** de su juego.
    - **Button:** `Name: btnCargar`, `Text: "Cargar Partida"`
        - **Descripción:** Botón para ejecutar una acción.
        - **Propiedades y Utilización:**
            - **`Name` (btnCargar):** Identificador único.
            - **`Text` ("Cargar Partida"):** Texto visible.
            - **Evento `Click`:** Se activa al hacer clic. Este evento **lee el archivo "partida.txt"**, recupera el índice de la pregunta guardada, lo convierte a entero (`CInt()`), y luego **actualiza la `lblPregunta.Text`** para mostrar la pregunta correspondiente, reanudando la partida. También verifica si el archivo existe.
        - **Uso en el programa:** Permite al usuario **continuar un juego guardado** previamente.
    - **Label:** `Name: lblPregunta`
        - **Descripción:** Un control Label muestra texto.
        - **Propiedades y Utilización:**
            - **`Name` (lblPregunta):** Identificador único.
            - **`Text`:** Almacena la cadena de texto que se muestra. En este programa, `lblPregunta.Text` se actualiza para **mostrar la pregunta correcta** una vez que la partida ha sido cargada.
        - **Uso en el programa:** Muestra la pregunta en la que se reanuda el juego.

Programa 199. Juego de preguntas con dificultad progresiva

- **En qué consiste el ejercicio:** Este programa implementa un **juego de preguntas y respuestas que aumenta su dificultad a medida que el jugador avanza de nivel**. El juego comienza con preguntas fáciles y, al completar cada nivel, presenta preguntas más complejas.

- **Controles necesarios y su utilización:**
    - **Label:** `Name: lblPregunta`
        - **Descripción:** Un control Label muestra texto.
        - **Propiedades y Utilización:**
            - **`Name` (lblPregunta):** Identificador único.
            - **`Text`:** Almacena y muestra la pregunta actual. Su contenido **cambia dinámicamente según el nivel de dificultad**. Al inicio, muestra la primera pregunta del nivel "Fácil".
        - **Uso en el programa:** Presenta las preguntas al jugador.
    - **TextBox:** `Name: txtRespuesta`
        - **Descripción:** Campo de entrada de texto.
        - **Propiedades y Utilización:**
            - **`Name` (txtRespuesta):** Identificador único.
            - **`Text`:** Captura la respuesta del usuario. Se limpia (`txtRespuesta.Clear()`) después de cada verificación para preparar la siguiente entrada.
        - **Uso en el programa:** Permite al jugador introducir su respuesta.
    - **Button:** `Name: btnVerificar`, `Text: "Verificar"`
        - **Descripción:** Botón para ejecutar una acción.
        - **Propiedades y Utilización:**
            - **`Name` (btnVerificar):** Identificador único.
            - **`Text` ("Verificar"):** Texto visible.
            - **Evento `Click`:** Se activa al hacer clic. En este evento, el programa (después de una posible verificación de la respuesta, no mostrada en el extracto) **avanza al siguiente nivel de dificultad**. Dependiendo del `nivel` actual, actualiza `lblPregunta.Text` con la primera pregunta del próximo nivel (`preguntasMedio`, `preguntasDificil`) y **incrementa la variable `nivel`**.
        - **Uso en el programa:** Inicia el paso a la siguiente pregunta y al siguiente nivel de dificultad.

Programa 200. Cliente de correo con múltiples cuentas

- **En qué consiste el ejercicio:** Este programa es una **simulación conceptual** de un cliente de correo electrónico que permite al usuario seleccionar entre **múltiples cuentas** para enviar un correo. No realiza el envío real, sino que indica desde qué cuenta se "enviaría" el mensaje.
- **Controles necesarios y su utilización:**
    - **ComboBox:** `Name: cmbCuentas`, `Items: "Cuenta1", "Cuenta2"`
        - **Descripción:** Un control `ComboBox` muestra una lista desplegable de elementos entre los cuales el usuario puede seleccionar uno.
        - **Propiedades y Utilización:**

        - **`Name` (cmbCuentas):** Identificador único del control.
        - **`Items`:** Es una colección que contiene los elementos de la lista. En este caso, se asume que se pobló con "Cuenta1" y "Cuenta2".
        - **`SelectedItem`:** Devuelve el elemento actualmente seleccionado en el `ComboBox`. Se utiliza para **identificar la cuenta desde la cual se "enviará" el correo**.
    - **Uso en el programa:** Permite al usuario **elegir una de sus cuentas de correo** para simular el envío.
- **TextBox:** `Name: txtPara`
    - **Descripción:** Campo de entrada de texto.
    - **Propiedades y Utilización:**
        - **`Name` (txtPara):** Identificador único.
        - **`Text`:** Contiene la dirección de correo electrónico del destinatario ingresada por el usuario.
    - **Uso en el programa:** Para que el usuario **introduzca la dirección del destinatario**.
- **TextBox:** `Name: txtAsunto`
    - **Descripción:** Campo de entrada de texto.
    - **Propiedades y Utilización:**
        - **`Name` (txtAsunto):** Identificador único.
        - **`Text`:** Contiene el asunto del correo ingresado por el usuario.
    - **Uso en el programa:** Para que el usuario **escriba el asunto del correo electrónico**.
- **TextBox:** `Name: txtCuerpo`, `Multiline: True`
    - **Descripción:** Un `TextBox` configurado para aceptar múltiples líneas de texto.
    - **Propiedades y Utilización:**
        - **`Name` (txtCuerpo):** Identificador único.
        - **`Multiline` (True):** Permite que el `TextBox` muestre y acepte texto en varias líneas, ideal para el cuerpo de un mensaje.
        - **`Text`:** Contiene el cuerpo del mensaje ingresado por el usuario.
    - **Uso en el programa:** Para que el usuario **redacte el cuerpo del mensaje de correo electrónico**.
- **Button:** `Name: btnEnviar`, `Text: "Enviar"`
    - **Descripción:** Botón para ejecutar una acción.
    - **Propiedades y Utilización:**
        - **`Name` (btnEnviar):** Identificador único.
        - **`Text` ("Enviar"):** Texto visible.
        - **Evento `Click`:** Se activa al hacer clic. Al ser un programa conceptual, este evento **no envía un correo real**, sino que muestra un `MessageBox` informando al usuario **desde qué cuenta se "enviaría" el correo y a quién**.

- **Uso en el programa:** Simula la acción de enviar un correo, mostrando los detalles de la cuenta y el destinatario.

Aquí tienes un manual detallado para los ejercicios del Programa 201 al 210, explicando en qué consiste cada uno y comentando ampliamente los controles y sus propiedades:

Programa 201. Cliente de correo con filtros de mensajes

- **En qué consiste el ejercicio:** Este programa **simula la aplicación de un filtro de texto a una lista de correos**. Permite al usuario introducir una cadena de texto y luego busca y muestra aquellos correos de una lista predefinida (o cargada) que contienen dicha cadena.
- **Controles necesarios y su utilización:**
  - **TextBox:** `Name: txtFiltro`
    - **Descripción:** Un campo de entrada de texto donde el usuario escribe el criterio de búsqueda o filtrado.
    - **Propiedades y Utilización:**
      - **`Name` (txtFiltro):** Es el identificador único del control en el código. Se utiliza para referirse a este TextBox en el código VB.NET.
      - **`Text`:** Almacena el contenido de texto que el usuario ha escrito. `txtFiltro.Text.ToLower()` se usa para obtener el texto del filtro y convertirlo a minúsculas, facilitando una búsqueda que no distinga entre mayúsculas y minúsculas.
    - **Uso en el programa:** Permite al usuario **introducir el texto** que desea usar para filtrar los correos.
  - **Button:** `Name: btnAplicarFiltro`, `Text: "Aplicar Filtro"`
    - **Descripción:** Un control Button es un elemento interactivo que ejecuta una acción cuando se hace clic en él.
    - **Propiedades y Utilización:**
      - **`Name` (btnAplicarFiltro):** Identificador único del botón.
      - **`Text` ("Aplicar Filtro"):** Es el texto que aparece visiblemente en el botón, indicando su función.
      - **Evento `Click`:** Es el evento principal que se activa cuando el usuario hace clic en el botón. Dentro de este evento, se implementa la lógica para iterar sobre la lista de correos y aplicar el filtro.
    - **Uso en el programa:** Al hacer clic, **inicia el proceso de filtrado** de los mensajes en `lstCorreos` según el texto de `txtFiltro`.
  - **ListBox:** `Name: lstCorreos`
    - **Descripción:** Un control ListBox muestra una lista de elementos. En este contexto, se asume que contiene los correos electrónicos a filtrar.
    - **Propiedades y Utilización:**
      - **`Name` (lstCorreos):** Identificador único del control.

        - **`Items`:** Es una colección que contiene todos los elementos que se muestran en el ListBox. En el código, se itera sobre `lstCorreos.Items` para comparar cada elemento con el filtro.
    - **Uso en el programa:** Se utiliza para **contener y recorrer los mensajes de correo** a los que se aplicará el filtro. Los mensajes que coinciden se muestran en un `MessageBox`.

Programa 202. Cliente de correo con carpetas personalizadas

- **En qué consiste el ejercicio:** Este programa permite al usuario **crear nuevas carpetas personalizadas** para organizar correos. Estas carpetas se visualizan en un control `TreeView`, que es ideal para estructuras jerárquicas.
- **Controles necesarios y su utilización:**
    - **TreeView:** `Name: treeCarpetas`
        - **Descripción:** Un control TreeView muestra una jerarquía de nodos, similar a la estructura de carpetas en un explorador de archivos.
        - **Propiedades y Utilización:**
            - **`Name` (treeCarpetas):** Identificador único del control.
            - **`Nodes`:** Es la colección principal de nodos en el nivel raíz del TreeView.
                - **`treeCarpetas.Nodes.Add(texto)`:** Se utiliza para **añadir un nuevo nodo** (representando una carpeta) al TreeView, usando el texto proporcionado como su nombre visible.
        - **Uso en el programa:** Muestra la **estructura jerárquica de las carpetas** creadas por el usuario.
    - **ListBox:** `Name: lstCorreos`
        - **Descripción:** Un control ListBox muestra una lista de elementos.
        - **Propiedades y Utilización:**
            - **`Name` (lstCorreos):** Identificador único del control.
        - **Uso en el programa:** Aunque el código proporcionado solo se centra en la creación de carpetas, se asume que `lstCorreos` **mostraría los correos electrónicos** dentro de las carpetas seleccionadas en `treeCarpetas`.
    - **Button:** `Name: btnCrearCarpeta`, `Text: "Crear Carpeta"`
        - **Descripción:** Un control Button para ejecutar una acción.
        - **Propiedades y Utilización:**
            - **`Name` (btnCrearCarpeta):** Identificador único del botón.
            - **`Text` ("Crear Carpeta"):** Texto visible en el botón.
            - **Evento `Click`:** Se activa al hacer clic. Este evento toma el texto de `txtNuevaCarpeta` y lo añade como un nuevo nodo al `treeCarpetas`.

        - **Uso en el programa:** Permite al usuario **confirmar la creación de una nueva carpeta**.
    - **TextBox:** `Name: txtNuevaCarpeta`
        - **Descripción:** Campo de entrada de texto donde el usuario escribe el nombre de la nueva carpeta.
        - **Propiedades y Utilización:**
            - **`Name` (txtNuevaCarpeta):** Identificador único.
            - **`Text`:** Contiene el nombre de la carpeta ingresado por el usuario. `txtNuevaCarpeta.Clear()` se usa para limpiar el campo después de que la carpeta ha sido creada.
        - **Uso en el programa:** Para que el usuario **escriba el nombre de la carpeta** que desea crear.

Programa 203. Cliente de correo con notificaciones

- **En qué consiste el ejercicio:** Este programa utiliza el control `NotifyIcon` para mostrar **notificaciones emergentes en la bandeja del sistema** (system tray). En este caso, simula una alerta sobre la llegada de un nuevo mensaje de correo.
- **Controles necesarios y su utilización:**
    - **NotifyIcon:** `Name: NotifyIcon1`, `Icon: icono.ico`
        - **Descripción:** Un control no visual que permite que la aplicación muestre un ícono en la bandeja del sistema de Windows. Es útil para notificaciones sin tener que mostrar la ventana principal de la aplicación.
        - **Propiedades y Utilización:**
            - **`Name` (NotifyIcon1):** Identificador único del control.
            - **`Icon` ("icono.ico"):** Establece el ícono que se mostrará en la bandeja del sistema. Este archivo de ícono (`.ico`) debe estar disponible para la aplicación.
            - **`Visible`:** Un valor booleano (`True`/`False`) que determina si el ícono de `NotifyIcon` es visible en la bandeja del sistema. `NotifyIcon1.Visible = True` asegura que el usuario vea el ícono antes de la notificación.
            - **`ShowBalloonTip(duration, title, text, icon)`:** Este método muestra la notificación en forma de globo.
                - **`duration` (3000):** El tiempo en milisegundos que la notificación permanecerá visible (3000 ms = 3 segundos).
                - **`title` ("Correo"):** El título que se mostrará en la parte superior del globo de notificación.
                - **`text` ("Tienes un nuevo mensaje"):** El mensaje principal que se mostrará dentro del globo de notificación.

        - **`icon` (ToolTipIcon.Info):** El tipo de ícono que se mostrará junto al mensaje, indicando el tipo de notificación (información, advertencia, error).
    - **Uso en el programa:** Permite mostrar **notificaciones discretas** sobre nuevos correos en la bandeja del sistema.
- **Button:** `Name: btnNotificar`, `Text: "Notificar"`
    - **Descripción:** Un control Button para ejecutar una acción.
    - **Propiedades y Utilización:**
        - **`Name` (btnNotificar):** Identificador único del botón.
        - **`Text` ("Notificar"):** Texto visible en el botón.
        - **Evento `Click`:** Se activa al hacer clic. Este evento activa la visibilidad del `NotifyIcon` y luego dispara el globo de notificación.
    - **Uso en el programa:** Permite al usuario **simular la recepción de un nuevo correo** y activar la notificación.

Programa 204. Cliente de correo con contactos

- **En qué consiste el ejercicio:** Este programa permite al usuario **agregar contactos** (nombre y dirección de correo electrónico) a una lista. Los contactos se muestran en un formato legible en un `ListBox`.
- **Controles necesarios y su utilización:**
    - **ListBox:** `Name: lstContactos`
        - **Descripción:** Un control ListBox muestra una lista de elementos.
        - **Propiedades y Utilización:**
            - **`Name` (lstContactos):** Identificador único del control.
            - **`Items`:** Colección de elementos que se muestran en el ListBox.
                - **`lstContactos.Items.Add(elemento)`:** Se utiliza para **añadir un nuevo contacto** (formateado como una cadena) a la lista visible.
        - **Uso en el programa: Muestra la lista de contactos** agregados por el usuario.
    - **TextBox:** `Name: txtNombre`
        - **Descripción:** Campo de entrada de texto para el nombre del contacto.
        - **Propiedades y Utilización:**
            - **`Name` (txtNombre):** Identificador único.
            - **`Text`:** Contiene el nombre del contacto. `txtNombre.Clear()` se usa para limpiar el campo después de agregar el contacto.
        - **Uso en el programa:** Para que el usuario **escriba el nombre** del contacto.
    - **TextBox:** `Name: txtCorreo`
        - **Descripción:** Campo de entrada de texto para la dirección de correo electrónico del contacto.

        - **Propiedades y Utilización:**
            - **`Name` (txtCorreo):** Identificador único.
            - **`Text`:** Contiene la dirección de correo. `txtCorreo.Clear()` se usa para limpiar el campo después de agregar el contacto.
        - **Uso en el programa:** Para que el usuario **escriba la dirección de correo** del contacto.
    - **Button:** `Name: btnAgregar`, `Text: "Agregar Contacto"`
        - **Descripción:** Botón para ejecutar una acción.
        - **Propiedades y Utilización:**
            - **`Name` (btnAgregar):** Identificador único.
            - **`Text` ("Agregar Contacto"):** Texto visible.
            - **Evento `Click`:** Se activa al hacer clic. Este evento concatena el nombre y el correo electrónico, y los añade a `lstContactos`.
        - **Uso en el programa: Confirma la adición del nuevo contacto** a la lista.

Programa 205. Cliente de correo con calendario

- **En qué consiste el ejercicio:** Este programa integra un control `MonthCalendar` que permite al usuario **seleccionar una fecha**. La fecha seleccionada se muestra en un `TextBox`.
- **Controles necesarios y su utilización:**
    - **MonthCalendar:** `Name: calendar`
        - **Descripción:** Un control visual que muestra un calendario de un mes a la vez. Permite al usuario seleccionar una fecha o un rango de fechas.
        - **Propiedades y Utilización:**
            - **`Name` (calendar):** Identificador único del control.
            - **`SelectionStart`:** Una propiedad que devuelve el objeto `DateTime` que representa la primera fecha seleccionada en el calendario. Si solo se selecciona una fecha, esta es la fecha elegida.
        - **Uso en el programa:** Permite al usuario **visualizar un calendario y seleccionar una fecha**.
    - **Button:** `Name: btnSeleccionar`, `Text: "Seleccionar Fecha"`
        - **Descripción:** Botón para ejecutar una acción.
        - **Propiedades y Utilización:**
            - **`Name` (btnSeleccionar):** Identificador único.
            - **`Text` ("Seleccionar Fecha"):** Texto visible.
            - **Evento `Click`:** Se activa al hacer clic. Este evento toma la fecha de `calendar.SelectionStart`, la convierte a un formato de cadena corta y la asigna al `txtFechaSeleccionada`.
        - **Uso en el programa:** Inicia la acción de **mostrar la fecha seleccionada** en el `TextBox`.
    - **TextBox:** `Name: txtFechaSeleccionada`

- **Descripción:** Campo de texto donde se muestra la fecha seleccionada.
- **Propiedades y Utilización:**
  - **`Name` (txtFechaSeleccionada):** Identificador único.
  - **`Text`:** Se actualiza programáticamente con la fecha elegida. `calendar.SelectionStart.ToShortDateString()` convierte la fecha a un formato corto (ej. "DD/MM/AAAA").
- **Uso en el programa: Muestra al usuario la fecha** que ha seleccionado en el calendario.

Programa 206. Cliente de correo con recordatorios

- **En qué consiste el ejercicio:** Este programa permite al usuario **establecer recordatorios con una fecha y un mensaje**. Los recordatorios se añaden a una lista visible en el formulario.
- **Controles necesarios y su utilización:**
  - **DateTimePicker:** `Name: dtpRecordatorio`
    - **Descripción:** Un control que permite al usuario seleccionar una fecha y/o hora de manera fácil y visual.
    - **Propiedades y Utilización:**
      - **`Name` (dtpRecordatorio):** Identificador único del control.
      - **`Value`:** Obtiene o establece el objeto `DateTime` que representa la fecha y hora actualmente seleccionadas.
      - **`dtpRecordatorio.Value.ToShortDateString()`:** Convierte la fecha seleccionada a un formato de cadena corta, ideal para mostrar en un `ListBox` junto con el mensaje.
    - **Uso en el programa:** Permite al usuario **seleccionar la fecha del recordatorio**.
  - **TextBox:** `Name: txtMensaje`
    - **Descripción:** Campo de entrada de texto para el mensaje del recordatorio.
    - **Propiedades y Utilización:**
      - **`Name` (txtMensaje):** Identificador único.
      - **`Text`:** Contiene el mensaje que el usuario desea asociar al recordatorio. `txtMensaje.Clear()` se usa para limpiar el campo después de agregar el recordatorio.
    - **Uso en el programa:** Para que el usuario **escriba el mensaje** del recordatorio.
  - **Button:** `Name: btnAgregar`, `Text: "Agregar Recordatorio"`
    - **Descripción:** Botón para ejecutar una acción.
    - **Propiedades y Utilización:**
      - **`Name` (btnAgregar):** Identificador único.
      - **`Text` ("Agregar Recordatorio"):** Texto visible.

        - **Evento `Click`:** Se activa al hacer clic. Este evento concatena el mensaje de `txtMensaje` con la fecha seleccionada de `dtpRecordatorio` y los añade a `lstRecordatorios`.
    - **Uso en el programa: Confirma la adición del nuevo recordatorio** a la lista.
- **ListBox:** `Name: lstRecordatorios`
    - **Descripción:** Muestra una lista de elementos.
    - **Propiedades y Utilización:**
        - **`Name` (lstRecordatorios):** Identificador único.
        - **`Items`:** Colección de elementos que se muestran en el ListBox.
            - **`lstRecordatorios.Items.Add(elemento)`:** Añade una cadena formateada con el mensaje y la fecha del recordatorio a la lista visible.
            - **`lstRecordatorios.Clear()`:** Se sugiere para limpiar el campo de texto, aunque la fuente indica `lstRecord` para esto. Asumo que es un error tipográfico y se refiere a `txtMensaje.Clear()`.
    - **Uso en el programa: Muestra la lista de recordatorios** que han sido agregados.

Programa 207. Cliente de correo con sincronización

- **En qué consiste el ejercicio:** Este programa es un **ejemplo conceptual** que simula el inicio de una sincronización de correos con un servidor. No implementa la lógica real de sincronización, sino que muestra un mensaje informativo.
- **Controles necesarios y su utilización:**
    - **Button:** `Name: btnSincronizar`, `Text: "Sincronizar"`
        - **Descripción:** Un control Button para ejecutar una acción.
        - **Propiedades y Utilización:**
            - **`Name` (btnSincronizar):** Identificador único del botón.
            - **`Text` ("Sincronizar"):** Texto visible en el botón.
            - **Evento `Click`:** Se activa al hacer clic. Este evento muestra un `MessageBox` informando al usuario que la sincronización se ha iniciado y aclara que una implementación real requeriría la interacción con un servidor de correo.
        - **Uso en el programa:** Simula la **acción de iniciar una sincronización** de correos.
    - **ListBox:** `Name: lstCorreos`
        - **Descripción:** Muestra una lista de elementos.
        - **Propiedades y Utilización:**
            - **`Name` (lstCorreos):** Identificador único del control.

- **Uso en el programa:** Aunque no se utiliza directamente en el código proporcionado, se asume que `lstCorreos` **mostraría los correos sincronizados** en una implementación completa del programa.

Programa 208. Cliente de correo con cifrado

- **En qué consiste el ejercicio:** Este programa permite al usuario **cifrar el cuerpo de un mensaje de correo** utilizando la codificación **Base64**. El texto original se introduce en un `TextBox` y el resultado cifrado se muestra en otro `TextBox`.
- **Controles necesarios y su utilización:**
  - **TextBox:** `Name: txtCuerpo`, `Multiline: True`
    - **Descripción:** Un campo de entrada de texto que permite múltiples líneas de texto, ideal para el cuerpo de un mensaje.
    - **Propiedades y Utilización:**
      - **`Name` (txtCuerpo):** Identificador único.
      - **`Multiline` (True):** Permite que el TextBox muestre y acepte texto en varias líneas.
      - **`Text`:** Almacena el texto plano que el usuario desea cifrar. `txtCuerpo.Text` se utiliza para obtener el contenido a codificar.
    - **Uso en el programa:** Para que el usuario **escriba el mensaje original** que desea cifrar.
  - **Button:** `Name: btnCifrar`, `Text: "Cifrar"`
    - **Descripción:** Botón para ejecutar una acción.
    - **Propiedades y Utilización:**
      - **`Name` (btnCifrar):** Identificador único.
      - **`Text` ("Cifrar"):** Texto visible.
      - **Evento `Click`:** Se activa al hacer clic. Este evento toma el texto de `txtCuerpo`, lo convierte a bytes UTF-8 y luego lo codifica a Base64, mostrando el resultado en `txtCifrado`.
    - **Uso en el programa:** Inicia el proceso de **cifrado del texto** ingresado.
  - **TextBox:** `Name: txtCifrado`, `Multiline: True`
    - **Descripción:** Un campo de texto que muestra el resultado del cifrado.
    - **Propiedades y Utilización:**
      - **`Name` (txtCifrado):** Identificador único.
      - **`Multiline` (True):** Permite mostrar texto en varias líneas, ya que el resultado Base64 puede ser largo.
      - **`Text`:** Se actualiza programáticamente con la cadena Base64 resultante del cifrado. `txtCifrado.Text = ...` se usa para mostrar el mensaje cifrado.
    - **Uso en el programa: Muestra el mensaje** una vez que ha sido cifrado con Base64.

Programa 209. Cliente de correo con firmas personalizadas

- **En qué consiste el ejercicio:** Este programa permite al usuario **añadir una firma predefinida** (o ingresada por él) al final del cuerpo de un mensaje de correo electrónico. El resultado se muestra en un `MessageBox` como el mensaje final listo para enviar.
- **Controles necesarios y su utilización:**
    - **TextBox:** `Name: txtCuerpo`, `Multiline: True`
        - **Descripción:** Un campo de entrada de texto que permite múltiples líneas, para el cuerpo principal del mensaje.
        - **Propiedades y Utilización:**
            - **`Name` (txtCuerpo):** Identificador único.
            - **`Multiline` (True):** Permite escribir un mensaje de varias líneas.
            - **`Text`:** Contiene el cuerpo del mensaje redactado por el usuario.
        - **Uso en el programa:** Para que el usuario **escriba el contenido principal** de su correo.
    - **TextBox:** `Name: txtFirma`
        - **Descripción:** Campo de entrada de texto para la firma del usuario.
        - **Propiedades y Utilización:**
            - **`Name` (txtFirma):** Identificador único.
            - **`Text`:** Contiene el texto de la firma que se añadirá al mensaje.
        - **Uso en el programa:** Para que el usuario **escriba su firma**.
    - **Button:** `Name: btnEnviar`, `Text: "Enviar"`
        - **Descripción:** Botón para ejecutar una acción.
        - **Propiedades y Utilización:**
            - **`Name` (btnEnviar):** Identificador único.
            - **`Text` ("Enviar"):** Texto visible.
            - **Evento `Click`:** Se activa al hacer clic. Este evento concatena el contenido de `txtCuerpo`, un salto de línea y el texto de `txtFirma` para formar el `mensajeFinal`, que luego se muestra en un `MessageBox`.
        - **Uso en el programa:** Simula la acción de **enviar un correo**, mostrando el mensaje completo con la firma.

Programa 210. Sistema de login con logs de actividad

- **En qué consiste el ejercicio:** Este programa implementa un sistema de login básico que **registra cada intento de inicio de sesión en un archivo de log** de actividad. El log incluye la fecha, hora y el usuario que intentó iniciar sesión.
- **Controles necesarios y su utilización:**
    - **TextBox:** `Name: txtUsuario`
        - **Descripción:** Campo de entrada de texto para el nombre de usuario.

        - **Propiedades y Utilización:**
            - **`Name` (txtUsuario):** Identificador único.
            - **`Text`:** Contiene el nombre de usuario ingresado. Se utiliza para registrar quién intenta iniciar sesión en el archivo de log.
        - **Uso en el programa:** Para que el usuario **introduzca su nombre de usuario**.
    - **TextBox:** `Name: txtContraseña`, `PasswordChar: *`
        - **Descripción:** Campo de entrada de texto diseñado para contraseñas.
        - **Propiedades y Utilización:**
            - **`Name` (txtContraseña):** Identificador único.
            - **`PasswordChar` (`"*"`):** Cuando se establece, cualquier carácter que el usuario escriba en el TextBox se muestra como el carácter especificado (un asterisco `*`), ocultando la contraseña para mayor seguridad.
            - **`Text`:** Contiene la contraseña ingresada. En este programa conceptual, se verifica que no esté vacía.
        - **Uso en el programa:** Para que el usuario **ingrese su contraseña de forma segura**.
    - **Button:** `Name: btnLogin`, `Text: "Iniciar Sesión"`
        - **Descripción:** Botón para ejecutar una acción.
        - **Propiedades y Utilización:**
            - **`Name` (btnLogin):** Identificador único.
            - **`Text` ("Iniciar Sesión"):** Texto visible.
            - **Evento `Click`:** Se activa al hacer clic. Este evento verifica si los campos de usuario y contraseña están vacíos. Si no lo están, registra el intento de inicio de sesión en el archivo de log y muestra un mensaje de bienvenida.
        - **Uso en el programa:** Inicia el proceso de **intento de inicio de sesión** y el registro de la actividad.

Programa 211. Sistema de login con auditoría

- **¿En qué consiste el ejercicio?** Este programa implementa un sistema de login que **registra cada intento de inicio de sesión en un control ListBox para fines de auditoría**. Indica la fecha, hora y el usuario que intentó acceder. El programa se enfoca en el registro del intento de acceso, no en la validación real de las credenciales.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtUsuario`
        - **Utilización:** Se usa para que el usuario ingrese su **nombre de usuario**.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre de usuario que el usuario ha introducido. En este programa, el contenido de `txtUsuario.Text` se integra en la cadena de auditoría.

    - **TextBox:**
        - **Name:** `txtContraseña`
        - **PasswordChar:** `*`
        - **Utilización:** Se usa para que el usuario ingrese su **contraseña**. La propiedad `PasswordChar` oculta los caracteres que el usuario escribe, reemplazándolos por un asterisco, lo cual es una práctica estándar de seguridad para evitar que la contraseña sea visible a terceros.
        - **Propiedades y su uso:** La propiedad `Text` contendrá la contraseña ingresada. Aunque no se usa directamente para la autenticación en este programa, es fundamental para la experiencia del usuario y la simulación de un login.
    - **Button:**
        - **Name:** `btnLogin`
        - **Text:** `"Iniciar Sesión"`
        - **Utilización:** Es el botón que el usuario presiona para **intentar iniciar sesión** y desencadenar el registro de auditoría.
        - **Propiedades y su uso:** El evento `Click` del botón (`btnLogin_Click`) es el disparador principal del programa. Al hacer clic, se ejecuta el código que construye el mensaje de auditoría y lo añade al ListBox. La propiedad `Text` (`"Iniciar Sesión"`) proporciona una etiqueta clara para la acción del botón.
    - **ListBox:**
        - **Name:** `lstAuditoria`
        - **Utilización:** Se usa para **mostrar y almacenar los registros de auditoría** de cada intento de inicio de sesión.
        - **Propiedades y su uso:** La colección `Items` del ListBox es crucial; `lstAuditoria.Items.Add(auditoria)` se utiliza para añadir una nueva entrada de texto a la lista. Cada entrada en esta lista representa un evento de auditoría.
- **Explicación:** Cuando el usuario hace clic en el botón `btnLogin`, se activa el evento `btnLogin_Click`. Dentro de este evento, se construye una cadena llamada `auditoria`. Esta cadena incluye la **fecha y hora actuales** (obtenida mediante `DateTime.Now.ToString()`), el **nombre de usuario** ingresado en `txtUsuario.Text`, y una descripción de la acción: "intentó iniciar sesión". Una vez construida, esta cadena de `auditoria` se añade como un nuevo elemento a la colección `Items` del `lstAuditoria`, haciendo visible el registro en el formulario. Finalmente, se muestra un `MessageBox` al usuario notificando que el intento ha sido registrado. La particularidad de este programa es que no realiza una autenticación de credenciales real, sino que se enfoca puramente en la función de registrar los intentos.

Programa 212. Sistema de login con autenticación externa

- **¿En qué consiste el ejercicio?** Este programa **simula un proceso de autenticación contra un servicio o sistema externo**. Aunque el código proporcionado realiza una validación local

simple, el concepto es que en una aplicación real, las credenciales se verificarían llamando a una API, un servidor LDAP o un proveedor de identidad.

- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtUsuario`
        - **Utilización:** Campo de texto donde el usuario introduce su **nombre de usuario**.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener la entrada del usuario, la cual se compara con un valor predefinido ("admin") en la simulación de autenticación.
    - **TextBox:**
        - **Name:** `txtContraseña`
        - **PasswordChar:** `*`
        - **Utilización:** Campo de texto donde el usuario introduce su **contraseña**. La propiedad `PasswordChar` oculta el texto escrito.
        - **Propiedades y su uso:** La propiedad `Text` se usa para obtener la contraseña ingresada, que en la simulación se compara con "1234".
    - **Button:**
        - **Name:** `btnLogin`
        - **Text:** `"Iniciar Sesión"`
        - **Utilización:** El botón que el usuario presiona para **realizar el intento de autenticación**.
        - **Propiedades y su uso:** El evento `Click` (`btnLogin_Click`) es el punto donde se ejecuta la lógica de autenticación simulada. La propiedad `Text` sirve como etiqueta de acción.
- **Explicación:** Al hacer clic en el botón `btnLogin`, se ejecuta el evento `btnLogin_Click`. El código dentro de este evento simula un proceso de autenticación externa. Compara el texto ingresado en `txtUsuario` con la cadena `"admin"` y el texto de `txtContraseña` con `"1234"`. Si ambas condiciones se cumplen (es decir, el usuario es "admin" y la contraseña es "1234"), se considera que la autenticación fue exitosa y se muestra un `MessageBox` con el mensaje **"Autenticación exitosa (externa)"**. En caso contrario, si las credenciales no coinciden con los valores predefinidos, se muestra un `MessageBox` con el mensaje **"Autenticación fallida"**. La nota clave de este programa es que, aunque realiza una verificación simple, está diseñado para conceptualizar la idea de que la validación de credenciales se realizaría normalmente contra un sistema ajeno a la aplicación local.

Programa 213. Sistema de login con validación de IP

- **¿En qué consiste el ejercicio?** Este programa implementa una **capa de seguridad adicional al verificar la dirección IP** del usuario que intenta iniciar sesión. El acceso se permite **solo si la dirección IP actual coincide con una IP autorizada** predefinida.
- **Controles necesarios en el formulario:**

    - **TextBox:**
        - **Name:** `txtUsuario`
        - **Utilización:** Campo para la entrada del **nombre de usuario**. Su valor no se usa en la lógica de validación de IP en este programa, pero es parte de la interfaz de un login típico.
        - **Propiedades y su uso:** La propiedad `Text` contiene el nombre de usuario.
    - **TextBox:**
        - **Name:** `txtContraseña`
        - **PasswordChar:** `*`
        - **Utilización:** Campo para la entrada de la **contraseña**, con caracteres ocultos. Su valor tampoco se usa en la lógica de validación de IP en este programa.
        - **Propiedades y su uso:** La propiedad `Text` contiene la contraseña. `PasswordChar` oculta la entrada.
    - **Button:**
        - **Name:** `btnLogin`
        - **Text:** `"Iniciar Sesión"`
        - **Utilización:** El botón que el usuario presiona para **intentar iniciar sesión** y desencadenar la validación de IP.
        - **Propiedades y su uso:** El evento `Click` (`btnLogin_Click`) es donde se ejecuta la lógica de verificación de IP. La propiedad `Text` etiqueta el botón.
- **Explicación:** Cuando se hace clic en el botón `btnLogin`, se activa el evento `btnLogin_Click`. Dentro de este evento, el programa define dos variables de cadena: `ipActual` y `ipPermitida`. Ambas están inicializadas con la dirección IP "192.168.1.100", **simulando que la IP actual del usuario es la misma que una IP autorizada**. El programa entonces compara estas dos cadenas. Si `ipActual` es igual a `ipPermitida`, se considera que la IP es válida y se muestra un `MessageBox` con el mensaje **"IP válida, acceso permitido"**. Si no coinciden, se muestra un `MessageBox` indicando **"IP no permitida"**. Es importante destacar que, en una aplicación real, la `ipActual` no sería un valor fijo, sino que se obtendría dinámicamente de la configuración de red del usuario o de una solicitud HTTP, y la `ipPermitida` provendría de una configuración de seguridad o una base de datos.

Programa 214. Sistema de login con tiempo de expiración

- **¿En qué consiste el ejercicio?** Este programa simula un **tiempo de expiración de sesión**. El acceso se concede **solo si el intento de login ocurre antes de una fecha y hora límite predefinidas**.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtUsuario`
        - **Utilización:** Campo para la entrada del **nombre de usuario**. Su valor no se usa en la lógica de expiración de tiempo, pero es parte de la interfaz de un login.

        - **Propiedades y su uso:** La propiedad `Text` contiene el nombre de usuario.
    - **TextBox:**
        - **Name:** `txtContraseña`
        - **PasswordChar:** `*`
        - **Utilización:** Campo para la entrada de la **contraseña**, con caracteres ocultos. Su valor tampoco se usa en la lógica de expiración de tiempo.
        - **Propiedades y su uso:** La propiedad `Text` contiene la contraseña. `PasswordChar` oculta la entrada.
    - **Button:**
        - **Name:** `btnLogin`
        - **Text:** `"Iniciar Sesión"`
        - **Utilización:** El botón que el usuario presiona para **intentar iniciar sesión** y desencadenar la verificación del tiempo de expiración.
        - **Propiedades y su uso:** El evento `Click` (`btnLogin_Click`) es donde se ejecuta la lógica de tiempo. La propiedad `Text` etiqueta el botón.
- **Explicación:** Se declara una variable a nivel de clase llamada `tiempoExpiracion` de tipo `DateTime`. Esta variable se inicializa en el momento en que se carga la clase, estableciendo su valor como la **hora actual más 30 minutos** (`DateTime.Now.AddMinutes(30)`). Esto define un plazo de 30 minutos desde el inicio del programa para que los logins sean válidos. Cuando el usuario hace clic en el botón `btnLogin`, se ejecuta el evento `btnLogin_Click`. El código compara la `DateTime.Now` (la fecha y hora actuales del sistema) con el valor de `tiempoExpiracion`.
    - Si `DateTime.Now` es **menor o igual** a `tiempoExpiracion`, se considera que la sesión no ha expirado y se muestra un `MessageBox` con el mensaje **"Sesión iniciada correctamente"**.
    - Si `DateTime.Now` es **mayor** que `tiempoExpiracion`, significa que el plazo de 30 minutos ha transcurrido, y se muestra un `MessageBox` con el mensaje **"Tiempo de sesión expirado"**. Este programa ofrece un ejemplo simplificado de cómo se podría gestionar una sesión de corta duración o una licencia temporal basada en el tiempo.

Programa 215. Sistema de login con recordatorio de sesión

- **¿En qué consiste el ejercicio?** Este programa utiliza un control `Timer` para **mostrar un recordatorio periódico al usuario después de iniciar sesión**, instándolo a cerrar la sesión si ha terminado su actividad.
- **Controles necesarios en el formulario:**
    - **Timer:**
        - **Name:** `Timer1`
        - **Interval:** `60000` (1 minuto)

        - **Utilización:** Este control se encarga de **ejecutar una acción repetidamente** después de un intervalo de tiempo específico. En este caso, su función principal es disparar el evento `Tick` cada minuto para mostrar el recordatorio.
        - **Propiedades y su uso:** La propiedad `Interval` se establece en `60000` milisegundos, lo que equivale a 1 minuto. Cuando el `Timer` se inicia (con `Timer1.Start()`), su evento `Tick` se dispara cada 60 segundos.
    - **Label:**
        - **Name:** `lblRecordatorio`
        - **Utilización:** Se usa para **mostrar el mensaje de recordatorio** al usuario.
        - **Propiedades y su uso:** La propiedad `Text` de este Label se actualiza cada vez que el `Timer1_Tick` se dispara, mostrando el mensaje "Recuerda cerrar sesión si terminas".
    - **TextBox:**
        - **Name:** `txtUsuario`
        - **Utilización:** Campo para la entrada del **nombre de usuario**. Se incluye para simular un proceso de login completo, aunque su valor no afecta la lógica del recordatorio.
        - **Propiedades y su uso:** La propiedad `Text` contiene el nombre de usuario.
    - **TextBox:**
        - **Name:** `txtContraseña`
        - **PasswordChar:** *
        - **Utilización:** Campo para la entrada de la **contraseña**, con caracteres ocultos. Similar a `txtUsuario`, su valor no influye en la lógica del recordatorio.
        - **Propiedades y su uso:** La propiedad `Text` contiene la contraseña. `PasswordChar` oculta la entrada.
    - **Button:**
        - **Name:** `btnLogin`
        - **Text:** `"Iniciar Sesión"`
        - **Utilización:** El botón que el usuario presiona para **iniciar la sesión** y, consecuentemente, activar el temporizador del recordatorio.
        - **Propiedades y su uso:** El evento `Click` (`btnLogin_Click`) se encarga de iniciar el `Timer1` con `Timer1.Start()` una vez que se "inicia la sesión". La propiedad `Text` etiqueta el botón.
- **Explicación:** La lógica principal de este programa se divide en dos partes: el inicio de sesión y el recordatorio periódico. Cuando el usuario hace clic en el botón `btnLogin`, se ejecuta el evento `btnLogin_Click`. Dentro de este evento, se llama a `Timer1.Start()` para **iniciar el temporizador**. Además, se muestra un `MessageBox` indicando "Sesión iniciada". Una vez que el `Timer1` está activo, su evento `Timer1_Tick` se disparará cada 60 segundos (debido a su propiedad `Interval` de 60000 milisegundos). Cada vez que este evento se activa, la propiedad `Text` del `lblRecordatorio` se actualiza para mostrar el mensaje **"Recuerda cerrar sesión si terminas"**. De esta manera, el usuario recibe un recordatorio constante, sin interrumpir su trabajo con ventanas emergentes, sino a través de un cambio en un Label en el formulario.

Programa 216. Sistema de login con restricción de horarios

- **¿En qué consiste el ejercicio?** Este programa implementa una medida de seguridad que **limita el acceso al sistema solo a un rango de horas predefinido**. Rechaza los intentos de inicio de sesión que ocurran fuera de ese horario establecido.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtUsuario`
        - **Utilización:** Campo para la entrada del **nombre de usuario**. No se utiliza en la lógica de restricción de horarios en este programa.
        - **Propiedades y su uso:** La propiedad `Text` contiene el nombre de usuario.
    - **TextBox:**
        - **Name:** `txtContraseña`
        - **PasswordChar:** `*`
        - **Utilización:** Campo para la entrada de la **contraseña**, con caracteres ocultos. Su valor tampoco se usa en la lógica de restricción de horarios.
        - **Propiedades y su uso:** La propiedad `Text` contiene la contraseña. `PasswordChar` oculta la entrada.
    - **Button:**
        - **Name:** `btnLogin`
        - **Text:** `"Iniciar Sesión"`
        - **Utilización:** El botón que el usuario presiona para **intentar iniciar sesión** y activar la verificación del horario.
        - **Propiedades y su uso:** El evento `Click` (`btnLogin_Click`) es donde se ejecuta la lógica de restricción de horarios. La propiedad `Text` etiqueta el botón.
- **Explicación:** Cuando se hace clic en el botón `btnLogin`, se activa el evento `btnLogin_Click`. Dentro de este evento, el programa obtiene la **hora actual del sistema** utilizando `DateTime.Now.TimeOfDay` y la almacena en una variable `horaActual` de tipo `TimeSpan`. Luego, se definen dos objetos `TimeSpan` adicionales para establecer el **rango de horario permitido**: `inicio` se establece a las 8:00:00 AM (`New TimeSpan(8, 0, 0)`) y `fin` se establece a las 6:00:00 PM (18:00:00) (`New TimeSpan(18, 0, 0)`). El programa compara la `horaActual` con este rango:
    - Si `horaActual` es **mayor o igual** que `inicio` **Y** `horaActual` es **menor o igual** que `fin`, se considera que el acceso está dentro del horario permitido, y se muestra un `MessageBox` con el mensaje **"Acceso permitido"**.
    - En cualquier otro caso (si la hora actual está fuera de este rango), se muestra un `MessageBox` con el mensaje **"Acceso fuera de horario permitido"**. Este programa es útil para implementar políticas de seguridad que restringen el acceso a sistemas o funcionalidades durante ciertas horas del día, por ejemplo, para tareas de mantenimiento o para cumplir con normativas empresariales.

Programa 217. Sistema de login con acceso remoto

- **¿En qué consiste el ejercicio?** Este programa es un **ejemplo conceptual** que indica cómo se gestionaría un acceso remoto. Destaca la **necesidad de una infraestructura de servidor** y código complejo para una implementación real de acceso remoto.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtUsuario`
        - **Utilización:** Campo para la entrada del **nombre de usuario**. Se incluye para simular un proceso de login típico.
        - **Propiedades y su uso:** La propiedad `Text` contendría el nombre de usuario.
    - **TextBox:**
        - **Name:** `txtContraseña`
        - **PasswordChar:** `*`
        - **Utilización:** Campo para la entrada de la **contraseña**, con caracteres ocultos.
        - **Propiedades y su uso:** La propiedad `Text` contendría la contraseña. `PasswordChar` oculta la entrada.
    - **Button:**
        - **Name:** `btnLogin`
        - **Text:** `"Iniciar Sesión"`
        - **Utilización:** El botón que el usuario presiona para **simular el intento de inicio de sesión remoto**.
        - **Propiedades y su uso:** El evento `Click` (`btnLogin_Click`) es donde se ejecuta la lógica conceptual. La propiedad `Text` etiqueta el botón.
- **Explicación:** Al hacer clic en el botón `btnLogin`, se ejecuta el evento `btnLogin_Click`. El código dentro de este evento es muy simple y consiste en una única línea que muestra un `MessageBox`. Este `MessageBox` tiene el mensaje **"Acceso remoto verificado (implementación real requiere servidor)"**. La "Explicación" del programa subraya que este ejemplo es meramente conceptual. En una aplicación real, la verificación del acceso remoto implicaría:
    - **Conexión a un servidor remoto**.
    - **Envío seguro de credenciales** a través de la red.
    - **Validación de esas credenciales** en el servidor.
    - **Recepción de una respuesta** del servidor indicando éxito o fracaso. Este programa simplemente ilustra la idea sin implementar la complejidad subyacente de la comunicación de red.

Programa 218. Sistema de login con sincronización de datos

- **¿En qué consiste el ejercicio?** Este programa es un **ejemplo conceptual que simula la sincronización de datos con un servidor** después de un inicio de sesión exitoso.
- **Controles necesarios en el formulario:**

    - **TextBox:**
        - **Name:** `txtUsuario`
        - **Utilización:** Campo para la entrada del **nombre de usuario**. Parte de la interfaz de login, pero no utilizada en la lógica de sincronización.
        - **Propiedades y su uso:** La propiedad `Text` contendría el nombre de usuario.
    - **TextBox:**
        - **Name:** `txtContraseña`
        - **PasswordChar:** `*`
        - **Utilización:** Campo para la entrada de la **contraseña**, con caracteres ocultos.
        - **Propiedades y su uso:** La propiedad `Text` contendría la contraseña. `PasswordChar` oculta la entrada.
    - **Button:**
        - **Name:** `btnLogin`
        - **Text:** `"Iniciar Sesión"`
        - **Utilización:** El botón que el usuario presiona para **simular el inicio de sesión y la posterior sincronización de datos**.
        - **Propiedades y su uso:** El evento `Click` (`btnLogin_Click`) es donde se ejecuta la lógica conceptual. La propiedad `Text` etiqueta el botón.
- **Explicación:** Cuando el usuario hace clic en el botón `btnLogin`, se ejecuta el evento `btnLogin_Click`. El código dentro de este evento muestra un `MessageBox` con el mensaje conceptual **"Datos sincronizados con servidor al iniciar sesión"**. La "Explicación" del programa detalla que, en una aplicación real, este paso de "sincronización de datos" implicaría una serie de acciones significativas después de que el usuario haya iniciado sesión exitosamente:
    - **Llamadas a servicios web o APIs**: Para obtener o actualizar información específica del usuario desde un servidor.
    - **Interacción con bases de datos remotas**: Para cargar el perfil del usuario, configuraciones, preferencias, historial de actividad, etc.
    - **Descarga de archivos**: Si el usuario tiene archivos asociados a su cuenta que deben estar disponibles localmente. Este programa, sin embargo, se limita a comunicar el concepto de que la sincronización de datos es una operación común que ocurre después del login, sin implementar el complejo código de red o de base de datos necesario para ello.

Programa 219. Sistema de login con cifrado de base de datos

- **¿En qué consiste el ejercicio?** Este programa demuestra una **forma básica de cifrar una contraseña utilizando la codificación Base64** antes de que esta sea (conceptualmente) almacenada o comparada en una base de datos.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtUsuario`

        - **Utilización:** Campo para la entrada del **nombre de usuario**. Se incluye para simular un proceso de login.
        - **Propiedades y su uso:** La propiedad `Text` contendría el nombre de usuario.
    - **TextBox:**
        - **Name:** `txtContraseña`
        - **PasswordChar:** `*`
        - **Utilización:** Campo para la entrada de la **contraseña** del usuario, con caracteres ocultos. Esta es la contraseña que se cifrará.
        - **Propiedades y su uso:** La propiedad `Text` es fundamental, ya que contiene la cadena de texto de la contraseña que será procesada para el cifrado. `PasswordChar` oculta la entrada.
    - **Button:**
        - **Name:** `btnLogin`
        - **Text:** `"Iniciar Sesión"`
        - **Utilización:** El botón que el usuario presiona para **simular el proceso de login** y desencadenar el cifrado de la contraseña ingresada.
        - **Propiedades y su uso:** El evento `Click` (`btnLogin_Click`) es donde se implementa la lógica de cifrado. La propiedad `Text` etiqueta el botón.
- **Explicación:** Este programa requiere la importación del espacio de nombres `System.Text` para manejar las codificaciones. Cuando el usuario hace clic en el botón `btnLogin`, se activa el evento `btnLogin_Click`. Dentro de este evento, el texto ingresado en `txtContraseña.Text` se somete a un proceso de codificación Base64:
    - **`Encoding.UTF8.GetBytes(txtContraseña.Text)`**: Primero, la cadena de texto de la contraseña se convierte en una matriz de bytes utilizando la codificación UTF-8.
    - **`Convert.ToBase64String(...)`**: Luego, esta matriz de bytes se convierte en una cadena Base64. El resultado es una cadena `contrasenaCifrada` que representa la contraseña original en un formato codificado. Finalmente, un `MessageBox` muestra esta `contrasenaCifrada` al usuario. La explicación del programa hace una **advertencia crucial**: aunque Base64 se utiliza para "cifrar" en este ejemplo conceptual, **Base64 es una codificación, no un cifrado seguro para contraseñas**. Para la seguridad real de contraseñas en bases de datos, se deben utilizar funciones hash criptográficas (como SHA256, que se verá en el siguiente programa) junto con un "salt" para mayor robustez. Este programa sirve para ilustrar un concepto básico de transformación de datos antes del almacenamiento.

Programa 220. Sistema de login con hash de contraseñas

- **¿En qué consiste el ejercicio?** Este programa demuestra el **uso de una función hash criptográfica (SHA256) para procesar una contraseña**. Esto es una **práctica segura y recomendada para almacenar contraseñas** en bases de datos, en lugar de guardarlas en texto plano o con codificaciones débiles como Base64.
- **Controles necesarios en el formulario:**

- **TextBox:**
    - **Name:** `txtUsuario`
    - **Utilización:** Campo para la entrada del **nombre de usuario**. Se incluye para simular un proceso de login completo.
    - **Propiedades y su uso:** La propiedad `Text` contendría el nombre de usuario.
- **TextBox:**
    - **Name:** `txtContraseña`
    - **PasswordChar:** `*`
    - **Utilización:** Campo para la entrada de la **contraseña** del usuario, con caracteres ocultos. Esta es la contraseña que se convertirá en un hash.
    - **Propiedades y su uso:** La propiedad `Text` es fundamental, ya que contiene la cadena de texto de la contraseña que será sometida al proceso de hashing. `PasswordChar` oculta la entrada.
- **Button:**
    - **Name:** `btnLogin`
    - **Text:** `"Iniciar Sesión"`
    - **Utilización:** El botón que el usuario presiona para **simular el proceso de login** y desencadenar el cálculo del hash de la contraseña ingresada.
    - **Propiedades y su uso:** El evento `Click` (`btnLogin_Click`) es donde se implementa la lógica de hashing. La propiedad `Text` etiqueta el botón.

- **Explicación:** Este programa requiere la importación de dos espacios de nombres cruciales: `System.Security.Cryptography` (para las funciones hash) y `System.Text` (para el manejo de codificaciones). Cuando el usuario hace clic en el botón `btnLogin`, se activa el evento `btnLogin_Click`. Dentro de este evento, se lleva a cabo el proceso de hashing SHA256:
    - **`Dim sha As SHA256 = SHA256.Create()`**: Se crea una nueva instancia de la clase `SHA256`, que es el algoritmo de hashing que se utilizará.
    - **`Dim bytes As Byte() = Encoding.UTF8.GetBytes(txtContraseña.Text)`**: El texto de la contraseña ingresado en `txtContraseña.Text` se convierte en una matriz de bytes utilizando la codificación UTF-8. Las funciones hash operan sobre datos binarios.
    - **`Dim hash As Byte() = sha.ComputeHash(bytes)`**: La instancia de `SHA256` (`sha`) calcula el valor hash de la matriz de bytes de la contraseña. El resultado es otra matriz de bytes que representa el hash.
    - **`Dim hashString As String = BitConverter.ToString(hash).Replace("-", "")`**: La matriz de bytes del hash se convierte a una representación de cadena hexadecimal legible. El método `Replace("-", "")` elimina los guiones que `BitConverter.ToString` inserta por defecto, resultando en una cadena hexadecimal continua. Finalmente, un `MessageBox` muestra esta `hashString`, que es el **hash SHA256 de la contraseña**. La explicación enfatiza que este es el **método recomendado para almacenar contraseñas de forma segura**. Un hash es una función unidireccional; es decir, es fácil calcular el hash a partir de la contraseña,

pero prácticamente imposible revertir el hash para obtener la contraseña original. La recomendación también incluye el uso de un "salt" (una cadena aleatoria única para cada contraseña) junto con el hash para proteger aún más contra ataques como las tablas arcoíris.

Programa 221. Sistema de inventario básico

- **¿En qué consiste el ejercicio?** Este programa implementa un **sistema de inventario simple que permite al usuario agregar productos con su cantidad** a un control `ListBox` para su visualización.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtProducto`
        - **Utilización:** Se usa para que el usuario ingrese el **nombre del producto** que desea añadir al inventario.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del producto introducido. Después de agregar, `txtProducto.Clear()` se usa para limpiar el campo.
    - **TextBox:**
        - **Name:** `txtCantidad`
        - **Utilización:** Se usa para que el usuario ingrese la **cantidad del producto** a registrar.
        - **Propiedades y su uso:** La propiedad `Text` se usa para obtener la cantidad. Después de agregar, `txtCantidad.Clear()` se usa para limpiar el campo.
    - **Button:**
        - **Name:** `btnAgregar`
        - **Text:** `"Agregar"`
        - **Utilización:** Es el botón que el usuario presiona para **añadir el producto y su cantidad** al `ListBox`.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregar_Click`) es el disparador. Al hacer clic, se ejecuta el código que concatena la información y la añade al `ListBox`. La propiedad `Text` (`"Agregar"`) proporciona una etiqueta clara para la acción.
    - **ListBox:**
        - **Name:** `lstInventario`
        - **Utilización:** Se usa para **mostrar los productos y sus cantidades** que se han agregado al inventario.
        - **Propiedades y su uso:** La colección `Items` del `ListBox` es crucial. `lstInventario.Items.Add()` se utiliza para añadir una nueva entrada de texto, combinando el nombre del producto y su cantidad, por ejemplo, "Producto - Cantidad: X".

- **Explicación:** Cuando el usuario hace clic en el botón `btnAgregar`, se activa el evento `btnAgregar_Click`. Dentro de este evento, el programa toma el texto de `txtProducto` y `txtCantidad`, los combina en una cadena de texto descriptiva (por ejemplo, "Leche - Cantidad: 10"), y luego añade esta cadena como un nuevo elemento a la colección `Items` del `lstInventario`. Después de añadir el elemento, los campos `txtProducto` y `txtCantidad` se limpian automáticamente para facilitar la entrada de nuevos productos.

Programa 222. Sistema de inventario con base de datos

- **¿En qué consiste el ejercicio?** Este programa simula un sistema de inventario utilizando un **`DataTable` como una representación de una base de datos en memoria**, y muestra los productos y sus cantidades en un control `DataGridView`.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtProducto`
        - **Utilización:** Se usa para que el usuario introduzca el **nombre del producto** a registrar.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del producto. Después de añadir, `txtProducto.Clear()` limpia el campo.
    - **TextBox:**
        - **Name:** `txtCantidad`
        - **Utilización:** Se usa para que el usuario introduzca la **cantidad del producto**.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener la cantidad. Después de añadir, `txtCantidad.Clear()` limpia el campo.
    - **Button:**
        - **Name:** `btnAgregar`
        - **Text:** `"Agregar"`
        - **Utilización:** Es el botón para **añadir el producto y su cantidad** al `DataTable` y, por ende, al `DataGridView`.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregar_Click`) se encarga de crear una nueva fila en el `DataTable` con los datos de los `TextBox`.
    - **DataGridView:**
        - **Name:** `dgvInventario`
        - **Utilización:** Se usa para **visualizar los productos y sus cantidades** en un formato tabular, enlazado directamente a los datos del `DataTable`.
        - **Propiedades y su uso:** La propiedad `DataSource` es clave; `dgvInventario.DataSource = dt` vincula el control al `DataTable` `dt`, haciendo que cualquier cambio en `dt` se refleje automáticamente en el

`DataGridView`. Las columnas "Producto" y "Cantidad" se definen programáticamente en el `Form1_Load`.

- **Explicación:** Al inicio del programa (en el evento `Form1_Load`), se crea un objeto `DataTable` llamado `dt` y se le definen dos columnas: "Producto" y "Cantidad". Luego, el `dgvInventario` se enlaza a este `DataTable` mediante su propiedad `DataSource`. Cuando el usuario hace clic en `btnAgregar`, el evento `btnAgregar_Click` añade una nueva fila al `dt.Rows` utilizando el texto de `txtProducto` y `txtCantidad`. Debido al enlace de datos (`DataSource`), el `DataGridView` se actualiza automáticamente para mostrar la nueva entrada. Los `TextBox` se limpian para la siguiente entrada.

Programa 223. Sistema de inventario con facturación

- **¿En qué consiste el ejercicio?** Este programa permite calcular el **costo total de un producto (cantidad por precio)** y añade esta información, junto con el nombre del producto, a una lista de facturación en un `ListBox`.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtProducto`
        - **Utilización:** Campo para el **nombre del producto**.
        - **Propiedades y su uso:** La propiedad `Text` contiene el nombre. Se limpia con `txtProducto.Clear()` después de añadir.
    - **TextBox:**
        - **Name:** `txtCantidad`
        - **Utilización:** Campo para la **cantidad del producto**.
        - **Propiedades y su uso:** `Text` obtiene la cantidad, y `Val()` la convierte a un valor numérico para la multiplicación. Se limpia con `txtCantidad.Clear()`.
    - **TextBox:**
        - **Name:** `txtPrecio`
        - **Utilización:** Campo para el **precio unitario del producto**.
        - **Propiedades y su uso:** `Text` obtiene el precio, y `Val()` la convierte a un valor numérico. Se limpia con `txtPrecio.Clear()`.
    - **Button:**
        - **Name:** `btnAgregar`
        - **Text:** `"Agregar"`
        - **Utilización:** Botón para **añadir el producto facturado** a la lista.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregar_Click`) realiza el cálculo y la adición a la lista.
    - **ListBox:**
        - **Name:** `lstFactura`
        - **Utilización:** Muestra los **ítems de la factura**, incluyendo producto, cantidad y total calculado.

        - **Propiedades y su uso:** `lstFactura.Items.Add()` añade una cadena formateada con "Producto - Cantidad: X - Total: Y".
- **Explicación:** Cuando el usuario hace clic en `btnAgregar`, el evento `btnAgregar_Click` se activa. Primero, los valores de `txtCantidad.Text` y `txtPrecio.Text` se convierten a números usando `Val()` y se multiplican para obtener el `total`. Luego, una cadena de texto se forma combinando el `txtProducto.Text`, la cantidad del `txtCantidad.Text`, y el `total` calculado, y esta cadena se añade como un nuevo elemento a `lstFactura.Items`. Finalmente, los tres `TextBox` (`txtProducto`, `txtCantidad`, `txtPrecio`) se limpian para facilitar la entrada de nuevas facturas.

Programa 224. Sistema de inventario con reportes

- **¿En qué consiste el ejercicio?** Este programa genera un **reporte simple del inventario** al recopilar todos los elementos actualmente listados en un `ListBox` y mostrarlos en un cuadro de mensaje (`MessageBox`).
- **Controles necesarios en el formulario:**
    - **Button:**
        - **Name:** `btnReporte`
        - **Text:** `"Generar Reporte"`
        - **Utilización:** Es el botón que el usuario presiona para **crear y visualizar el reporte** del inventario.
        - **Propiedades y su uso:** El evento `Click` (`btnReporte_Click`) desencadena la recopilación de los datos y la construcción del mensaje del reporte.
    - **ListBox:**
        - **Name:** `lstInventario`
        - **Utilización:** Contiene la **lista de productos y sus cantidades** que conforman el inventario y serán incluidos en el reporte.
        - **Propiedades y su uso:** La colección `Items` se itera (`For Each item In lstInventario.Items`) para obtener cada elemento del inventario. `item.ToString()` convierte cada elemento a su representación de cadena para ser incluido en el reporte.
- **Explicación:** Al hacer clic en el botón `btnReporte`, se ejecuta el evento `btnReporte_Click`. El código inicializa una cadena `reporte` con un título "Reporte Inventario:" y un salto de línea (`Environment.NewLine`). Luego, un bucle `For Each` recorre cada `item` en `lstInventario.Items`. En cada iteración, el `item` se convierte a cadena y se concatena al `reporte`, seguido de otro salto de línea. Una vez que el bucle ha procesado todos los elementos, el `reporte` completo se muestra al usuario en un `MessageBox`.

Programa 225. Sistema de inventario con gráficos

- **¿En qué consiste el ejercicio?** Este programa visualiza datos de inventario simulados mediante un **gráfico de columnas** que se muestra en un control `Chart`.
- **Controles necesarios en el formulario:**
    - **Chart:**
        - **Name:** `chartInventario`
        - **ChartType:** `Column` (configurado en tiempo de ejecución o diseño)
        - **Utilización:** Control gráfico que **muestra visualmente los datos** de inventario.
        - **Propiedades y su uso:** La propiedad `Series` es una colección donde se añaden los conjuntos de datos. `chartInventario.Series.Clear()` limpia series anteriores. `ChartType` de una serie se puede establecer a `DataVisualization.Charting.SeriesChartType.Column` para columnas. `Points.AddXY()` añade puntos de datos al gráfico, especificando una etiqueta X y un valor Y.
    - **Button:**
        - **Name:** `btnGraficar`
        - **Text:** `"Mostrar Gráfico"`
        - **Utilización:** Botón para **generar y actualizar el gráfico** de inventario.
        - **Propiedades y su uso:** El evento `Click` (`btnGraficar_Click`) es el que contiene la lógica para construir y mostrar el gráfico.
- **Explicación:** Al hacer clic en el botón `btnGraficar`, se activa el evento `btnGraficar_Click`. El código comienza limpiando cualquier serie de datos existente en `chartInventario` con `chartInventario.Series.Clear()`. Luego, crea una nueva instancia de `Series` (una colección de puntos de datos) y la nombra "Inventario". Establece el `ChartType` de esta serie como `Column` para que se muestre como un gráfico de columnas. A continuación, añade dos puntos de datos simulados: "Producto A" con un valor de 10, y "Producto B" con un valor de 15, usando `serie.Points.AddXY()`. Finalmente, esta serie `serie` se añade a la colección `chartInventario.Series`, lo que hace que el gráfico se dibuje y se muestre en el formulario. **Requisito adicional:** Para usar el control `Chart`, es necesario asegurarse de que la referencia `System.Windows.Forms.DataVisualization` esté añadida al proyecto.

Programa 226. Sistema de inventario con exportación Excel

- **¿En qué consiste el ejercicio?** Este programa es un **ejemplo conceptual que simula la exportación de datos de inventario a una hoja de cálculo de Microsoft Excel**. No contiene una implementación real de la exportación, sino que indica lo que se necesitaría.
- **Controles necesarios en el formulario:**
    - **Button:**
        - **Name:** `btnExportar`

        - **Text:** `"Exportar a Excel"`
        - **Utilización:** Botón que el usuario presiona para **iniciar el proceso conceptual de exportación**.
        - **Propiedades y su uso:** El evento `Click` (`btnExportar_Click`) es el único controlador de eventos en este programa, y simplemente muestra un mensaje.
- **Explicación:** Cuando el usuario hace clic en el botón `btnExportar`, se ejecuta el evento `btnExportar_Click`. El código dentro de este evento es muy simple y consiste en una única línea que muestra un `MessageBox` con el mensaje: **"Datos exportados a Excel (requiere referencia a Microsoft Excel)"**. **Requisito adicional:** La fuente aclara que, para una implementación real de esta funcionalidad, se necesitaría agregar una referencia a la librería de objetos de Microsoft Excel en el proyecto. Este programa se limita a ilustrar el concepto sin el código complejo de interacción con Excel.

Programa 227. Sistema de inventario con importación Excel

- **¿En qué consiste el ejercicio?** Este programa es un **ejemplo conceptual que simula la importación de datos de inventario desde una hoja de cálculo de Microsoft Excel**. Al igual que el de exportación, se centra en la idea más que en la implementación completa.
- **Controles necesarios en el formulario:**
    - **Button:**
        - **Name:** `btnImportar`
        - **Text:** `"Importar desde Excel"`
        - **Utilización:** Botón que el usuario presiona para **iniciar el proceso conceptual de importación**.
        - **Propiedades y su uso:** El evento `Click` (`btnImportar_Click`) es el único controlador de eventos y solo muestra un mensaje informativo.
- **Explicación:** Al hacer clic en el botón `btnImportar`, se activa el evento `btnImportar_Click`. El código consiste en una línea que muestra un `MessageBox` con el mensaje: **"Datos importados desde Excel (requiere referencia a Microsoft Excel)"**. **Requisito adicional:** Similar al programa de exportación, la fuente indica que para una implementación funcional, se requeriría agregar una referencia a la librería de objetos de Microsoft Excel en el proyecto. El programa, por lo tanto, es un marcador de posición para una funcionalidad más compleja.

Programa 228. Sistema de inventario con códigos de barra

- **¿En qué consiste el ejercicio?** Este programa permite al usuario **agregar "códigos de barra" (tratados como texto simple)** a una lista de inventario que se muestra en un `ListBox`.
- **Controles necesarios en el formulario:**
    - **TextBox:**

        - **Name:** `txtCodigo`
        - **Utilización:** Se usa para que el usuario ingrese el **código de barra** (como texto).
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener la entrada del usuario. Después de agregar, `txtCodigo.Clear()` limpia el campo para la siguiente entrada.
    - **Button:**
        - **Name:** `btnAgregar`
        - **Text:** `"Agregar Código"`
        - **Utilización:** Es el botón que el usuario presiona para **añadir el código ingresado** a la lista.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregar_Click`) se encarga de añadir el texto del `TextBox` al `ListBox`.
    - **ListBox:**
        - **Name:** `lstCodigos`
        - **Utilización:** Se usa para **mostrar la lista de códigos de barra** que se han registrado.
        - **Propiedades y su uso:** `lstCodigos.Items.Add()` se utiliza para añadir el texto directamente desde `txtCodigo.Text` como un nuevo elemento a la lista.
- **Explicación:** Cuando el usuario hace clic en el botón `btnAgregar`, se activa el evento `btnAgregar_Click`. El programa toma el texto ingresado en `txtCodigo.Text` y lo añade directamente como un nuevo elemento a la colección `Items` del `lstCodigos`. Inmediatamente después, el campo `txtCodigo` se vacía utilizando `txtCodigo.Clear()` para que el usuario pueda ingresar otro código.

Programa 229. Sistema de inventario con múltiples usuarios

- **¿En qué consiste el ejercicio?** Este programa simula el **registro de acciones de inventario (agregar un producto con cantidad) realizadas por diferentes tipos de usuarios**, cuya identidad es seleccionada previamente desde un `ComboBox`.
- **Controles necesarios en el formulario:**
    - **ComboBox:**
        - **Name:** `cmbUsuarios`
        - **Items:** `"Admin"`, `"Vendedor"` (se asume que se añaden en el diseñador o en `Form_Load`)
        - **Utilización:** Permite al usuario **seleccionar el rol o tipo de usuario** que está realizando la acción de inventario.
        - **Propiedades y su uso:** La propiedad `SelectedItem` de `cmbUsuarios` se utiliza para obtener la cadena que representa el usuario actual (`"Admin"` o `"Vendedor"`).
    - **TextBox:**
        - **Name:** `txtProducto`

        - **Utilización:** Campo para el **nombre del producto** que se está agregando.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del producto.
    - **TextBox:**
        - **Name:** `txtCantidad`
        - **Utilización:** Campo para la **cantidad del producto**.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener la cantidad.
    - **Button:**
        - **Name:** `btnAgregar`
        - **Text:** `"Agregar"`
        - **Utilización:** Botón para **simular la adición de un producto** y registrar quién la realizó.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregar_Click`) es donde se recoge la información del usuario y del producto para mostrar el mensaje de simulación.
- **Explicación:** Al hacer clic en el botón `btnAgregar`, se activa el evento `btnAgregar_Click`. El código obtiene la cadena del elemento seleccionado en `cmbUsuarios` (que representa el tipo de usuario), el texto de `txtProducto` y el texto de `txtCantidad`. Luego, muestra un `MessageBox` que simula la acción, indicando **"Usuario: [Tipo de Usuario] agregó [Nombre Producto] Cantidad: [Cantidad]"**. Este programa ilustra cómo se podría registrar una acción con el contexto de un usuario específico, sin implicar una lógica de backend o almacenamiento real.

Programa 230. Sistema de inventario con permisos

- **¿En qué consiste el ejercicio?** Este programa demuestra cómo la **acción de "eliminar producto" puede estar restringida a ciertos roles de usuario**, como "Admin", basándose en la selección de un `ComboBox`.
- **Controles necesarios en el formulario:**
    - **ComboBox:**
        - **Name:** `cmbRoles`
        - **Items:** `"Admin"`, `"Vendedor"` (se asume que se añaden en el diseñador o en `Form_Load`)
        - **Utilización:** Permite al usuario **seleccionar el rol** que está intentando realizar la acción de eliminación.
        - **Propiedades y su uso:** La propiedad `SelectedItem` se utiliza para obtener la cadena del rol seleccionado y compararla con `"Admin"`.
    - **Button:**
        - **Name:** `btnEliminar`
        - **Text:** `"Eliminar Producto"`
        - **Utilización:** Botón que el usuario presiona para **simular la eliminación de un producto**.

        - **Propiedades y su uso:** El evento `Click` (`btnEliminar_Click`) contiene la lógica de verificación de permisos y muestra un mensaje de resultado.
- **Explicación:** Cuando el usuario hace clic en el botón `btnEliminar`, se activa el evento `btnEliminar_Click`. El código verifica la cadena del `cmbRoles.SelectedItem`.
    - Si el rol seleccionado es **"Admin"**, se muestra un `MessageBox` con el mensaje **"Producto eliminado"**, simulando que la acción se ha realizado con éxito.
    - Si el rol seleccionado **no es "Admin"** (por ejemplo, "Vendedor"), se muestra un `MessageBox` con el mensaje **"No tiene permisos para eliminar"**, indicando la restricción. Este programa es un ejemplo básico de control de acceso basado en roles.

Programa 231. Sistema de inventario con auditoría

- **¿En qué consiste el ejercicio?** Este programa implementa un sistema de inventario que **registra las adiciones de productos en un control ListBox específico para auditoría**, incluyendo la fecha y hora del evento.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtProducto`
        - **Utilización:** Se usa para que el usuario ingrese el **nombre del producto** que desea añadir al inventario.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del producto introducido. Después de agregar, `txtProducto.Clear()` se usa para limpiar el campo.
    - **TextBox:**
        - **Name:** `txtCantidad`
        - **Utilización:** Se usa para que el usuario ingrese la **cantidad del producto** a registrar.
        - **Propiedades y su uso:** La propiedad `Text` se usa para obtener la cantidad. Después de agregar, `txtCantidad.Clear()` se usa para limpiar el campo.
    - **Button:**
        - **Name:** `btnAgregar`
        - **Text:** `"Agregar"`
        - **Utilización:** Es el botón que el usuario presiona para **añadir el producto y su cantidad** al inventario y registrar la acción.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregar_Click`) es el disparador. Al hacer clic, se ejecuta el código que concatena la información y la añade a ambos `ListBox`. La propiedad `Text` (`"Agregar"`) proporciona una etiqueta clara para la acción.
    - **ListBox:**
        - **Name:** `lstInventario`
        - **Utilización:** Se usa para **mostrar los productos y sus cantidades** que se han agregado al inventario.

        - **Propiedades y su uso:** La colección `Items` del `ListBox` es donde se añade la entrada principal del inventario, con `lstInventario.Items.Add()`.
    - **ListBox:**
        - **Name:** `lstAuditoria`
        - **Utilización:** Se usa para **mostrar los registros de auditoría** de cada adición de producto.
        - **Propiedades y su uso:** La colección `Items` de este `ListBox` se utiliza para añadir una nueva entrada de auditoría formateada con la fecha, hora y el nombre del producto, usando `lstAuditoria.Items.Add()`.
- **Explicación:** Cuando el usuario hace clic en el botón `btnAgregar`, se activa el evento `btnAgregar_Click`. Dentro de este evento, el programa realiza dos acciones principales: primero, añade una cadena combinando el nombre del producto (`txtProducto.Text`) y su cantidad (`txtCantidad.Text`) a `lstInventario`. Segundo, registra una entrada en `lstAuditoria` que incluye la **fecha y hora actuales** (obtenida mediante `DateTime.Now.ToString()`) y una descripción del evento: "Producto agregado: [nombre del producto]". Después de estas operaciones, los campos `txtProducto` y `txtCantidad` se limpian para la siguiente entrada.

Programa 232. Sistema de inventario con estadísticas

- **¿En qué consiste el ejercicio?** Este programa proporciona una **estadística básica del inventario** al mostrar el número total de productos (elementos) actualmente registrados en el `ListBox` mediante un `MessageBox`.
- **Controles necesarios en el formulario:**
    - **Button:**
        - **Name:** `btnEstadisticas`
        - **Text:** `"Mostrar Estadísticas"`
        - **Utilización:** Es el botón que el usuario presiona para **obtener y visualizar las estadísticas** del inventario.
        - **Propiedades y su uso:** El evento `Click` (`btnEstadisticas_Click`) es el disparador. Al hacer clic, se ejecuta el código que calcula y muestra el total de productos. La propiedad `Text` etiqueta el botón.
    - **ListBox:**
        - **Name:** `lstInventario`
        - **Utilización:** Contiene la **lista de productos** del inventario. Aunque no se visualiza directamente en la acción del botón, es la fuente de datos para la estadística.
        - **Propiedades y su uso:** La propiedad `Items.Count` de `lstInventario` es crucial, ya que devuelve el número de elementos que hay en la lista, lo que se interpreta como el total de productos.
- **Explicación:** Cuando el usuario hace clic en el botón `btnEstadisticas`, se activa el evento `btnEstadisticas_Click`. Dentro de este evento, el programa obtiene el número de elementos presentes en `lstInventario` utilizando

`lstInventario.Items.Count` y lo almacena en la variable `totalProductos`. Finalmente, un `MessageBox` se encarga de mostrar al usuario esta estadística, indicando "Total de productos en inventario: [número]".

Programa 233. Sistema de inventario con gráficos dinámicos

- **¿En qué consiste el ejercicio?** Este programa visualiza los productos que se encuentran en el `lstInventario` mediante un **gráfico de columnas dinámico** que se muestra en un control `Chart`. Cada producto se representa como una columna en el gráfico.
- **Controles necesarios en el formulario:**
    - **Chart:**
        - **Name:** `chartInventario`
        - **ChartType:** `Column` (configurado en tiempo de ejecución)
        - **Utilización:** Es el control gráfico que **muestra visualmente los datos** del inventario.
        - **Propiedades y su uso:** La propiedad `Series` es una colección donde se añaden los conjuntos de datos. `chartInventario.Series.Clear()` limpia series anteriores. `ChartType` de una serie se establece a `DataVisualization.Charting.SeriesChartType.Column` para que se muestre como un gráfico de columnas. `Points.AddXY(item.ToString(), 1)` añade puntos de datos, usando el nombre del producto como etiqueta X y un valor fijo de 1 para representar su existencia.
    - **Button:**
        - **Name:** `btnActualizarGrafico`
        - **Text:** `"Actualizar Gráfico"`
        - **Utilización:** Es el botón que el usuario presiona para **generar y actualizar el gráfico** con los datos actuales del `lstInventario`.
        - **Propiedades y su uso:** El evento `Click` (`btnActualizarGrafico_Click`) es el que contiene la lógica para construir y mostrar el gráfico.
    - **ListBox:**
        - **Name:** `lstInventario`
        - **Utilización:** Contiene la **lista de productos** que se utilizarán como datos para el gráfico.
        - **Propiedades y su uso:** La colección `Items` de `lstInventario` se itera (`For Each item In lstInventario.Items`) para obtener cada nombre de producto y añadirlo al gráfico.
- **Explicación:** Al hacer clic en el botón `btnActualizarGrafico`, se activa el evento `btnActualizarGrafico_Click`. El código comienza por **limpiar cualquier serie de datos existente** en `chartInventario` con `chartInventario.Series.Clear()`. Luego, se crea una nueva instancia de `Series` (que es una colección de puntos de datos) y se nombra "Inventario". Se establece el `ChartType` de esta serie como `Column`. A

continuación, un bucle `For Each` recorre cada `item` en la colección `lstInventario.Items`. Por cada `item`, se añade un punto al gráfico utilizando `serie.Points.AddXY(item.ToString(), 1)`. Esto significa que cada producto del `ListBox` aparecerá como una columna en el gráfico, con el nombre del producto como etiqueta y una altura de 1 (simbolizando su existencia). Finalmente, esta `serie` se añade a la colección `chartInventario.Series`, lo que hace que el gráfico se dibuje y se muestre en el formulario. **Requisito adicional:** Para utilizar el control `Chart`, es necesario que la referencia `System.Windows.Forms.DataVisualization` esté añadida al proyecto.

Programa 234. Sistema de inventario con múltiples almacenes

- **¿En qué consiste el ejercicio?** Este programa permite al usuario **agregar productos al inventario, especificando de qué almacén provienen** al seleccionarlo de un control `ComboBox`.
- **Controles necesarios en el formulario:**
    - **ComboBox:**
        - **Name:** `cmbAlmacenes`
        - **Items:** `"Almacén 1"`, `"Almacén 2"` (se asume que se añaden en el diseñador o en el evento `Form_Load`)
        - **Utilización:** Permite al usuario **seleccionar el almacén** al que pertenece el producto que se va a agregar.
        - **Propiedades y su uso:** La propiedad `SelectedItem` de `cmbAlmacenes` se utiliza para obtener la cadena que representa el almacén elegido por el usuario.
    - **TextBox:**
        - **Name:** `txtProducto`
        - **Utilización:** Campo de texto donde el usuario introduce el **nombre del producto**.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del producto. Después de añadir, `txtProducto.Clear()` limpia el campo.
    - **TextBox:**
        - **Name:** `txtCantidad`
        - **Utilización:** Campo de texto donde el usuario introduce la **cantidad del producto**.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener la cantidad. Después de añadir, `txtCantidad.Clear()` limpia el campo.
    - **Button:**
        - **Name:** `btnAgregar`
        - **Text:** `"Agregar"`
        - **Utilización:** El botón que el usuario presiona para **añadir la información del producto, cantidad y almacén** al `ListBox`.

        - **Propiedades y su uso:** El evento `Click` (`btnAgregar_Click`) es el que contiene la lógica para combinar y añadir los datos.
    - **ListBox:**
        - **Name:** `lstInventario`
        - **Utilización:** Se usa para **mostrar los productos registrados**, incluyendo el almacén de origen, el nombre y la cantidad.
        - **Propiedades y su uso:** `lstInventario.Items.Add()` se utiliza para añadir una cadena formateada con el almacén, producto y cantidad a la lista.
- **Explicación:** Cuando el usuario hace clic en el botón `btnAgregar`, se activa el evento `btnAgregar_Click`. El código obtiene el nombre del almacén seleccionado de `cmbAlmacenes.SelectedItem`. Luego, construye una cadena de texto combinando el nombre del almacén, el nombre del producto (`txtProducto.Text`) y la cantidad (`txtCantidad.Text`), con el formato "Almacén: [Nombre Almacén] - Producto: [Nombre Producto] - Cantidad: [Cantidad]". Esta cadena se añade como un nuevo elemento a `lstInventario.Items`. Finalmente, los campos `txtProducto` y `txtCantidad` se limpian para la siguiente entrada.

Programa 235. Sistema de inventario con conexión remota

- **¿En qué consiste el ejercicio?** Este programa es un **ejemplo conceptual que simula la conexión a un servidor remoto** para la gestión del inventario, sin implementar la complejidad real de una conexión de red.
- **Controles necesarios en el formulario:**
    - **Button:**
        - **Name:** `btnConectar`
        - **Text:** `"Conectar a Servidor"`
        - **Utilización:** Es el botón que el usuario presiona para **simular el intento de conexión** a un servidor remoto.
        - **Propiedades y su uso:** El evento `Click` (`btnConectar_Click`) es el único controlador de eventos en este programa, y simplemente muestra un mensaje.
- **Explicación:** Al hacer clic en el botón `btnConectar`, se ejecuta el evento `btnConectar_Click`. El código dentro de este evento es muy simple y consiste en una única línea que muestra un `MessageBox` con el mensaje: **"Conexión remota establecida (simulada)"**. La explicación del programa aclara que este es solo un concepto y que una implementación real de una conexión remota requeriría un código de red mucho más complejo, que no se incluye en este ejemplo.

Programa 236. Sistema de inventario con backup automático

- **¿En qué consiste el ejercicio?** Este programa es un **ejemplo conceptual que simula la realización de un backup automático** de los datos del inventario.

- **Controles necesarios en el formulario:**
    - **Button:**
        - **Name:** `btnBackup`
        - **Text:** `"Realizar Backup"`
        - **Utilización:** Es el botón que el usuario presiona para **simular la ejecución de un backup**.
        - **Propiedades y su uso:** El evento `Click` (`btnBackup_Click`) es el único controlador de eventos en este programa, y simplemente muestra un mensaje.
- **Explicación:** Al hacer clic en el botón `btnBackup`, se ejecuta el evento `btnBackup_Click`. El código dentro de este evento es una única línea que muestra un `MessageBox` con el mensaje: **"Backup automático realizado correctamente"**. El programa conceptualiza la idea de un backup automático, pero la implementación real implicaría operaciones como copiar archivos de bases de datos, exportar datos o utilizar herramientas específicas de backup, las cuales no están presentes en este ejemplo.

Programa 237. Sistema de inventario con restauración de datos

- **¿En qué consiste el ejercicio?** Este programa es un **ejemplo conceptual que simula la restauración de datos** del inventario a partir de un backup previamente realizado.
- **Controles necesarios en el formulario:**
    - **Button:**
        - **Name:** `btnRestaurar`
        - **Text:** `"Restaurar Backup"`
        - **Utilización:** Es el botón que el usuario presiona para **simular la restauración de los datos**.
        - **Propiedades y su uso:** El evento `Click` (`btnRestaurar_Click`) es el único controlador de eventos en este programa, y simplemente muestra un mensaje.
- **Explicación:** Cuando el usuario hace clic en el botón `btnRestaurar`, se ejecuta el evento `btnRestaurar_Click`. El código dentro de este evento es una única línea que muestra un `MessageBox` con el mensaje: **"Datos restaurados desde backup"**. Este programa ilustra conceptualmente la acción de restaurar datos, pero una implementación real requeriría operaciones más complejas como reemplazar archivos de bases de datos o importar datos de un archivo de backup anterior.

Programa 238. Sistema de inventario con notificaciones

- **¿En qué consiste el ejercicio?** Este programa utiliza un control `NotifyIcon` para **mostrar una notificación emergente en la bandeja del sistema (system tray)**, alertando al usuario sobre eventos importantes en el inventario, como un "stock bajo" de productos.
- **Controles necesarios en el formulario:**

- **NotifyIcon:**
    - **Name:** `NotifyIcon1`
    - **Icon:** `icono.ico` (se requiere un archivo de ícono)
    - **Utilización:** Es el control que permite a la aplicación mostrar un icono en la bandeja del sistema y **enviar notificaciones tipo "globo"** al usuario.
    - **Propiedades y su uso:** `NotifyIcon1.Visible = True` asegura que el icono de la aplicación esté visible en la bandeja del sistema. `NotifyIcon1.ShowBalloonTip()` es el método principal que muestra la notificación. Sus parámetros controlan la duración (3000 ms = 3 segundos), el título de la notificación ("Inventario"), el mensaje ("Alerta: Stock bajo en algunos productos") y el tipo de icono que acompaña la notificación (`ToolTipIcon.Warning` para indicar una alerta).
- **Button:**
    - **Name:** `btnNotificar`
    - **Text:** `"Notificar Stock Bajo"`
    - **Utilización:** Es el botón que el usuario presiona para **simular y activar la notificación** de stock bajo.
    - **Propiedades y su uso:** El evento `Click` (`btnNotificar_Click`) es el disparador que ejecuta la lógica para mostrar la notificación.
- **Explicación:** Al hacer clic en el botón `btnNotificar`, se activa el evento `btnNotificar_Click`. Dentro de este evento, la propiedad `NotifyIcon1.Visible` se establece en `True` para asegurar que el icono de la aplicación aparezca en la bandeja del sistema si aún no lo está. Luego, el método `NotifyIcon1.ShowBalloonTip()` se invoca para mostrar la notificación. Esta notificación tiene un título de "Inventario", un mensaje de "Alerta: Stock bajo en algunos productos", una duración de 3 segundos y utiliza un icono de advertencia (`ToolTipIcon.Warning`) para indicar la naturaleza urgente del mensaje. **Requisito adicional:** Para que `NotifyIcon1` funcione correctamente, se debe haber agregado un archivo de ícono (por ejemplo, `icono.ico`) a la propiedad `Icon` del control en tiempo de diseño.

Programa 239. Sistema de inventario con control de lotes

- **¿En qué consiste el ejercicio?** Este programa permite al usuario **agregar productos al inventario, incluyendo un número de lote** junto con el nombre del producto y la cantidad, y muestra esta información en un `ListBox`.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtProducto`
        - **Utilización:** Se usa para que el usuario ingrese el **nombre del producto**.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del producto. Después de agregar, `txtProducto.Clear()` se usa para limpiar el campo.
    - **TextBox:**

        - **Name:** `txtLote`
        - **Utilización:** Se usa para que el usuario ingrese el **número de lote** asociado al producto.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el número de lote. Después de agregar, `txtLote.Clear()` se usa para limpiar el campo.
    - **TextBox:**
        - **Name:** `txtCantidad`
        - **Utilización:** Se usa para que el usuario ingrese la **cantidad del producto**.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener la cantidad. Después de agregar, `txtCantidad.Clear()` se usa para limpiar el campo.
    - **Button:**
        - **Name:** `btnAgregar`
        - **Text:** `"Agregar Lote"`
        - **Utilización:** Es el botón que el usuario presiona para **añadir la información del producto, lote y cantidad** al inventario.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregar_Click`) es el disparador. Al hacer clic, se ejecuta el código que concatena la información y la añade al `ListBox`.
    - **ListBox:**
        - **Name:** `lstInventario`
        - **Utilización:** Se usa para **mostrar los productos registrados**, incluyendo el nombre, número de lote y cantidad.
        - **Propiedades y su uso:** `lstInventario.Items.Add()` se utiliza para añadir una cadena formateada con "Producto: [Nombre] - Lote: [Número] - Cantidad: [Cantidad]" a la lista.
- **Explicación:** Cuando el usuario hace clic en el botón `btnAgregar`, se activa el evento `btnAgregar_Click`. El programa toma el texto de `txtProducto`, `txtLote` y `txtCantidad`, los combina en una cadena de texto descriptiva (por ejemplo, "Producto: Leche - Lote: ABC123 - Cantidad: 10"), y luego añade esta cadena como un nuevo elemento a la colección `Items` del `lstInventario`. Después de añadir el elemento, los tres `TextBox` (`txtProducto`, `txtLote`, y `txtCantidad`) se limpian automáticamente para facilitar la entrada de nuevos productos.

Programa 240. Sistema de inventario con caducidades

- **¿En qué consiste el ejercicio?** Este programa permite al usuario **agregar productos al inventario junto con su cantidad y una fecha de caducidad**, la cual se selecciona mediante un control `DateTimePicker`.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtProducto`

        - **Utilización:** Se usa para que el usuario ingrese el **nombre del producto**.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del producto. Después de agregar, `txtProducto.Clear()` se usa para limpiar el campo.
    - **DateTimePicker:**
        - **Name:** `dtpCaducidad`
        - **Utilización:** Permite al usuario **seleccionar una fecha de caducidad** para el producto.
        - **Propiedades y su uso:** La propiedad `Value` obtiene la fecha y hora seleccionadas. `ToShortDateString()` se utiliza para formatear esta fecha a una cadena corta y legible (ej. "DD/MM/AAAA").
    - **TextBox:**
        - **Name:** `txtCantidad`
        - **Utilización:** Se usa para que el usuario ingrese la **cantidad del producto**.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener la cantidad. Después de agregar, `txtCantidad.Clear()` se usa para limpiar el campo.
    - **Button:**
        - **Name:** `btnAgregar`
        - **Text:** `"Agregar Producto"`
        - **Utilización:** Es el botón que el usuario presiona para **añadir el producto, su cantidad y la fecha de caducidad** al inventario.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregar_Click`) es el disparador. Al hacer clic, se ejecuta el código que concatena la información y la añade al `ListBox`.
    - **ListBox:**
        - **Name:** `lstInventario`
        - **Utilización:** Se usa para **mostrar los productos registrados**, incluyendo su nombre, cantidad y fecha de caducidad.
        - **Propiedades y su uso:** `lstInventario.Items.Add()` se utiliza para añadir una cadena formateada con "Producto: [Nombre] - Cantidad: [Cantidad] - Caduca: [Fecha]" a la lista.
- **Explicación:** Cuando el usuario hace clic en el botón `btnAgregar`, se activa el evento `btnAgregar_Click`. El programa obtiene el nombre del producto de `txtProducto.Text`, la cantidad de `txtCantidad.Text`, y la fecha seleccionada en `dtpCaducidad` (formateada como cadena corta con `dtpCaducidad.Value.ToShortDateString()`). Toda esta información se combina en una cadena de texto descriptiva (por ejemplo, "Producto: Manzanas - Cantidad: 50 - Caduca: 31/12/2024"). Esta cadena se añade como un nuevo elemento a la colección `Items` del `lstInventario`. Finalmente, los campos `txtProducto` y `txtCantidad` se limpian automáticamente para la siguiente entrada.

Programa 241. Sistema de ventas básico

- **¿En qué consiste el ejercicio?** Este programa implementa un **sistema de ventas simple que permite al usuario agregar productos con su cantidad** a un control `ListBox` para su visualización.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtProducto`
        - **Utilización:** Se usa para que el usuario ingrese el **nombre del producto** que desea añadir a la venta.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del producto introducido. Después de agregar la venta, `txtProducto.Clear()` se usa para limpiar el campo.
    - **TextBox:**
        - **Name:** `txtCantidad`
        - **Utilización:** Se usa para que el usuario ingrese la **cantidad del producto** a registrar en la venta.
        - **Propiedades y su uso:** La propiedad `Text` se usa para obtener la cantidad. Después de agregar la venta, `txtCantidad.Clear()` se usa para limpiar el campo.
    - **Button:**
        - **Name:** `btnAgregar`
        - **Text:** `"Agregar Venta"`
        - **Utilización:** Es el botón que el usuario presiona para **añadir el producto y su cantidad** a la lista de ventas.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregar_Click`) se activa al hacer clic en el botón. Este evento es el disparador para ejecutar el código que concatena la información y la añade al `ListBox`. La propiedad `Text` (`"Agregar Venta"`) proporciona una etiqueta clara para la acción del botón.
    - **ListBox:**
        - **Name:** `lstVentas`
        - **Utilización:** Se usa para **mostrar los productos y sus cantidades** que se han agregado a la lista de ventas.
        - **Propiedades y su uso:** La colección `Items` del `ListBox` es donde se añade la entrada de la venta. `lstVentas.Items.Add()` se utiliza para añadir una nueva entrada de texto, combinando el nombre del producto y su cantidad, por ejemplo, "Producto - Cantidad: X".
- **Explicación:** Cuando el usuario hace clic en el botón `btnAgregar`, se activa el evento `btnAgregar_Click`. Dentro de este evento, el programa toma el texto de `txtProducto` y `txtCantidad`, los concatena en una cadena de texto descriptiva (como "Producto - Cantidad: X"), y luego añade esta cadena como un nuevo elemento a la colección `Items` del `lstVentas`. Después de añadir el elemento, los campos

`txtProducto` y `txtCantidad` se limpian automáticamente para facilitar la entrada de nuevas ventas.

Programa 242. Sistema de ventas con base de datos

- **¿En qué consiste el ejercicio?** Este programa utiliza un **`DataTable` como una simulación de una base de datos en memoria** para almacenar productos y sus cantidades, y los muestra en un control `DataGridView`.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtProducto`
        - **Utilización:** Se usa para que el usuario introduzca el **nombre del producto** a registrar en la venta.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del producto. Después de añadir la venta, `txtProducto.Clear()` limpia el campo.
    - **TextBox:**
        - **Name:** `txtCantidad`
        - **Utilización:** Se usa para que el usuario introduzca la **cantidad del producto** a vender.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener la cantidad. Después de añadir la venta, `txtCantidad.Clear()` limpia el campo.
    - **Button:**
        - **Name:** `btnAgregar`
        - **Text:** `"Agregar Venta"`
        - **Utilización:** Es el botón para **añadir el producto y su cantidad** al `DataTable` y, por ende, al `DataGridView`.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregar_Click`) se encarga de crear una nueva fila en el `DataTable` con los datos de los `TextBox`.
    - **DataGridView:**
        - **Name:** `dgvVentas`
        - **Utilización:** Se usa para **visualizar los productos y sus cantidades** en un formato tabular, enlazado directamente a los datos del `DataTable`.
        - **Propiedades y su uso:** La propiedad `DataSource` es clave; `dgvVentas.DataSource = dt` vincula el control al `DataTable` `dt`, haciendo que cualquier cambio en `dt` se refleje automáticamente en el `DataGridView`. Las columnas "Producto" y "Cantidad" se definen programáticamente en el evento `Form1_Load`.
- **Explicación:** Al inicio del programa (en el evento `Form1_Load`), se declara un objeto `DataTable` llamado `dt` a nivel de clase. Dentro de `Form1_Load`, se le definen dos

columnas: "Producto" y "Cantidad". Luego, el `dgvVentas` se enlaza a este `DataTable` mediante su propiedad `DataSource`. Cuando el usuario hace clic en `btnAgregar`, el evento `btnAgregar_Click` añade una nueva fila a `dt.Rows` utilizando el texto de `txtProducto` y `txtCantidad`. Debido al enlace de datos (`DataSource`), el `DataGridView` se actualiza automáticamente para mostrar la nueva entrada. Los `TextBox` se limpian para la siguiente entrada.

Programa 243. Sistema de ventas con clientes

- **¿En qué consiste el ejercicio?** Este programa permite **asociar ventas a diferentes clientes** al seleccionarlos de un control `ComboBox`.
- **Controles necesarios en el formulario:**
    - **ComboBox:**
        - **Name:** `cmbClientes`
        - **Items:** `"Cliente 1"`, `"Cliente 2"` (se asume que se añaden en el diseñador o en un evento como `Form_Load`)
        - **Utilización:** Permite al usuario **seleccionar el cliente** al que se atribuye la venta.
        - **Propiedades y su uso:** La propiedad `SelectedItem` de `cmbClientes` se utiliza para obtener la cadena que representa el cliente elegido por el usuario.
    - **TextBox:**
        - **Name:** `txtProducto`
        - **Utilización:** Campo para el **nombre del producto** vendido.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del producto. Después de añadir, `txtProducto.Clear()` limpia el campo.
    - **TextBox:**
        - **Name:** `txtCantidad`
        - **Utilización:** Campo para la **cantidad del producto** vendido.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener la cantidad. Después de añadir, `txtCantidad.Clear()` limpia el campo.
    - **Button:**
        - **Name:** `btnAgregar`
        - **Text:** `"Agregar Venta"`
        - **Utilización:** El botón que el usuario presiona para **añadir la información de la venta** (cliente, producto y cantidad) al `ListBox`.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregar_Click`) es el que contiene la lógica para combinar y añadir los datos a la lista.
    - **ListBox:**
        - **Name:** `lstVentas`
        - **Utilización:** Muestra los **registros de ventas**, incluyendo el cliente, el producto y la cantidad.

        - **Propiedades y su uso:** `lstVentas.Items.Add()` se utiliza para añadir una cadena formateada con "Cliente: [Nombre Cliente] - Producto: [Nombre Producto] - Cantidad: [Cantidad]" a la lista.
- **Explicación:** Cuando el usuario hace clic en el botón `btnAgregar`, se activa el evento `btnAgregar_Click`. El código obtiene el nombre del cliente seleccionado de `cmbClientes.SelectedItem`. Luego, construye una cadena de texto combinando el nombre del cliente, el nombre del producto (`txtProducto.Text`) y la cantidad (`txtCantidad.Text`). Esta cadena se añade como un nuevo elemento a `lstVentas.Items`. Finalmente, los campos `txtProducto` y `txtCantidad` se limpian para la siguiente entrada.

Programa 244. Sistema de ventas con facturación

- **¿En qué consiste el ejercicio?** Este programa permite calcular el **costo total de una venta (cantidad por precio)** y añade esta información, junto con el nombre del producto, a una lista de facturación en un `ListBox`.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtProducto`
        - **Utilización:** Campo para el **nombre del producto** a facturar.
        - **Propiedades y su uso:** La propiedad `Text` contiene el nombre del producto. Se limpia con `txtProducto.Clear()` después de añadir la factura.
    - **TextBox:**
        - **Name:** `txtCantidad`
        - **Utilización:** Campo para la **cantidad del producto** a facturar.
        - **Propiedades y su uso:** `Text` obtiene la cantidad, y `Val()` la convierte a un valor numérico para la multiplicación. Se limpia con `txtCantidad.Clear()`.
    - **TextBox:**
        - **Name:** `txtPrecio`
        - **Utilización:** Campo para el **precio unitario del producto** a facturar.
        - **Propiedades y su uso:** `Text` obtiene el precio, y `Val()` la convierte a un valor numérico. Se limpia con `txtPrecio.Clear()`.
    - **Button:**
        - **Name:** `btnAgregar`
        - **Text:** `"Agregar Factura"`
        - **Utilización:** Botón para **añadir el producto facturado** a la lista.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregar_Click`) realiza el cálculo y la adición a la lista de facturas.
    - **ListBox:**
        - **Name:** `lstFactura`
        - **Utilización:** Muestra los **ítems de la factura**, incluyendo producto, cantidad y total calculado.

        - **Propiedades y su uso:** `lstFactura.Items.Add()` añade una cadena formateada con "Producto - Cantidad: X - Total: Y".
- **Explicación:** Cuando el usuario hace clic en `btnAgregar`, el evento `btnAgregar_Click` se activa. Primero, los valores de `txtCantidad.Text` y `txtPrecio.Text` se convierten a números usando `Val()` y se multiplican para obtener el `total`. Luego, una cadena de texto se forma combinando el `txtProducto.Text`, la cantidad del `txtCantidad.Text`, y el `total` calculado, y esta cadena se añade como un nuevo elemento a `lstFactura.Items`. Finalmente, los tres `TextBox` (`txtProducto`, `txtCantidad`, `txtPrecio`) se limpian para facilitar la entrada de nuevas facturas.

Programa 245. Sistema de ventas con reportes

- **¿En qué consiste el ejercicio?** Este programa genera un **reporte simple de las ventas** al recopilar todos los elementos actualmente listados en un `ListBox` y mostrarlos en un cuadro de mensaje (`MessageBox`).
- **Controles necesarios en el formulario:**
    - **Button:**
        - **Name:** `btnReporte`
        - **Text:** `"Generar Reporte"`
        - **Utilización:** Es el botón que el usuario presiona para **crear y visualizar el reporte** de las ventas.
        - **Propiedades y su uso:** El evento `Click` (`btnReporte_Click`) desencadena la recopilación de los datos y la construcción del mensaje del reporte.
    - **ListBox:**
        - **Name:** `lstVentas`
        - **Utilización:** Contiene la **lista de ventas** que conformarán el reporte.
        - **Propiedades y su uso:** La colección `Items` se itera (`For Each item In lstVentas.Items`) para obtener cada elemento de venta. `item.ToString()` convierte cada elemento a su representación de cadena para ser incluido en el reporte.
- **Explicación:** Al hacer clic en el botón `btnReporte`, se ejecuta el evento `btnReporte_Click`. El código inicializa una cadena `reporte` con un título "Reporte de Ventas:" y un salto de línea (`Environment.NewLine`). Luego, un bucle `For Each` recorre cada `item` en `lstVentas.Items`. En cada iteración, el `item` se convierte a cadena y se concatena al `reporte`, seguido de otro salto de línea. Una vez que el bucle ha procesado todos los elementos, el `reporte` completo se muestra al usuario en un `MessageBox`.

Programa 246. Sistema de ventas con estadísticas

- **¿En qué consiste el ejercicio?** Este programa proporciona una **estadística básica de las ventas** al mostrar el número total de ventas (elementos) actualmente registradas en el `ListBox` mediante un `MessageBox`.
- **Controles necesarios en el formulario:**
    - **Button:**
        - **Name:** `btnEstadisticas`
        - **Text:** `"Mostrar Estadísticas"`
        - **Utilización:** Es el botón que el usuario presiona para **obtener y visualizar las estadísticas** de las ventas.
        - **Propiedades y su uso:** El evento `Click` (`btnEstadisticas_Click`) es el disparador. Al hacer clic, se ejecuta el código que calcula y muestra el total de ventas. La propiedad `Text` etiqueta el botón.
    - **ListBox:**
        - **Name:** `lstVentas`
        - **Utilización:** Contiene la **lista de ventas** registradas. Es la fuente de datos para la estadística.
        - **Propiedades y su uso:** La propiedad `Items.Count` de `lstVentas` es crucial, ya que devuelve el número de elementos que hay en la lista, lo que se interpreta como el total de ventas.
- **Explicación:** Cuando el usuario hace clic en el botón `btnEstadisticas`, se activa el evento `btnEstadisticas_Click`. Dentro de este evento, el programa obtiene el número de elementos presentes en `lstVentas` utilizando `lstVentas.Items.Count` y lo almacena en la variable `totalVentas`. Finalmente, un `MessageBox` se encarga de mostrar al usuario esta estadística, indicando "Total de ventas registradas: [número]".

Programa 247. Sistema de ventas con gráficos

- **¿En qué consiste el ejercicio?** Este programa visualiza datos de ventas simulados mediante un **gráfico de columnas** que se muestra en un control `Chart`. Cada venta se representa como una columna en el gráfico.
- **Controles necesarios en el formulario:**
    - **Chart:**
        - **Name:** `chartVentas`
        - **ChartType:** `Column` (configurado en tiempo de ejecución)
        - **Utilización:** Control gráfico que **muestra visualmente los datos** de ventas.
        - **Propiedades y su uso:** La propiedad `Series` es una colección donde se añaden los conjuntos de datos. `chartVentas.Series.Clear()` limpia series anteriores. `ChartType` de una serie se puede establecer a `DataVisualization.Charting.SeriesChartType.Column` para columnas. `Points.AddXY()` añade puntos de datos al gráfico, especificando una etiqueta X y un valor Y.

    - **Button:**
        - **Name:** `btnGraficar`
        - **Text:** `"Mostrar Gráfico"`
        - **Utilización:** Botón para **generar y actualizar el gráfico** de ventas.
        - **Propiedades y su uso:** El evento `Click` (`btnGraficar_Click`) es el que contiene la lógica para construir y mostrar el gráfico.
    - **ListBox:**
        - **Name:** `lstVentas`
        - **Utilización:** Contiene la **lista de ventas** que se utilizarán como datos para el gráfico.
        - **Propiedades y su uso:** La colección `Items` de `lstVentas` se itera (`For Each item In lstVentas.Items`) para obtener cada elemento de venta y añadirlo al gráfico.
- **Explicación:** Al hacer clic en el botón `btnGraficar`, se activa el evento `btnGraficar_Click`. El código comienza limpiando cualquier serie de datos existente en `chartVentas` con `chartVentas.Series.Clear()`. Luego, crea una nueva instancia de `Series` (una colección de puntos de datos) y la nombra "Ventas". Establece el `ChartType` de esta serie como `Column` para que se muestre como un gráfico de columnas. A continuación, un bucle `For Each` recorre cada `item` en la colección `lstVentas.Items`. Por cada `item`, se añade un punto al gráfico utilizando `serie.Points.AddXY(item.ToString(), 1)`. Esto significa que cada venta del `ListBox` aparecerá como una columna en el gráfico, con el nombre de la venta como etiqueta y una altura de 1 (simbolizando su existencia). Finalmente, esta `serie` se añade a la colección `chartVentas.Series`, lo que hace que el gráfico se dibuje y se muestre en el formulario. **Requisito adicional:** Para usar el control `Chart`, es necesario asegurarse de que la referencia `System.Windows.Forms.DataVisualization` esté añadida al proyecto.

Programa 248. Sistema de ventas con exportación Excel

- **¿En qué consiste el ejercicio?** Este programa es un **ejemplo conceptual que simula la exportación de datos de ventas a una hoja de cálculo de Microsoft Excel**. No contiene una implementación real de la exportación, sino que indica lo que se necesitaría.
- **Controles necesarios en el formulario:**
    - **Button:**
        - **Name:** `btnExportar`
        - **Text:** `"Exportar a Excel"`
        - **Utilización:** Botón que el usuario presiona para **iniciar el proceso conceptual de exportación**.
        - **Propiedades y su uso:** El evento `Click` (`btnExportar_Click`) es el único controlador de eventos en este programa, y simplemente muestra un mensaje.

- **Explicación:** Cuando el usuario hace clic en el botón `btnExportar`, se ejecuta el evento `btnExportar_Click`. El código dentro de este evento es muy simple y consiste en una única línea que muestra un `MessageBox` con el mensaje: **"Ventas exportadas a Excel (requiere referencia a Microsoft Excel)"**. **Requisito adicional:** La fuente aclara que, para una implementación real de esta funcionalidad, se necesitaría agregar una referencia a la librería de objetos de Microsoft Excel en el proyecto. Este programa se limita a ilustrar el concepto sin el código complejo de interacción con Excel.

Programa 249. Sistema de ventas con importación Excel

- **¿En qué consiste el ejercicio?** Este programa es un **ejemplo conceptual que simula la importación de datos de ventas desde una hoja de cálculo de Microsoft Excel**. Al igual que el de exportación, se centra en la idea más que en la implementación completa.
- **Controles necesarios en el formulario:**
    - **Button:**
        - **Name:** `btnImportar`
        - **Text:** `"Importar desde Excel"`
        - **Utilización:** Botón que el usuario presiona para **iniciar el proceso conceptual de importación**.
        - **Propiedades y su uso:** El evento `Click` (`btnImportar_Click`) es el único controlador de eventos y solo muestra un mensaje informativo.
- **Explicación:** Al hacer clic en el botón `btnImportar`, se activa el evento `btnImportar_Click`. El código consiste en una línea que muestra un `MessageBox` con el mensaje: **"Ventas importadas desde Excel (requiere referencia a Microsoft Excel)"**. **Requisito adicional:** Similar al programa de exportación, la fuente indica que para una implementación funcional, se requeriría agregar una referencia a la librería de objetos de Microsoft Excel en el proyecto. El programa, por lo tanto, es un marcador de posición para una funcionalidad más compleja.

Programa 250. Sistema de ventas con múltiples usuarios

- **¿En qué consiste el ejercicio?** Este programa simula el **registro de ventas realizadas por diferentes tipos de usuarios**, cuya identidad es seleccionada previamente desde un `ComboBox`.
- **Controles necesarios en el formulario:**
    - **ComboBox:**
        - **Name:** `cmbUsuarios`
        - **Items:** `"Admin"`, `"Vendedor"` (se asume que se añaden en el diseñador o en un evento como `Form_Load`)
        - **Utilización:** Permite al usuario **seleccionar el rol o tipo de usuario** que está realizando la venta.

        - **Propiedades y su uso:** La propiedad `SelectedItem` de `cmbUsuarios` se utiliza para obtener la cadena que representa el usuario actual (`"Admin"` o `"Vendedor"`).
    - **TextBox:**
        - **Name:** `txtProducto`
        - **Utilización:** Campo para el **nombre del producto** que se está vendiendo.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del producto.
    - **TextBox:**
        - **Name:** `txtCantidad`
        - **Utilización:** Campo para la **cantidad del producto** vendido.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener la cantidad.
    - **Button:**
        - **Name:** `btnAgregar`
        - **Text:** `"Agregar Venta"`
        - **Utilización:** Botón para **simular la adición de una venta** y registrar quién la realizó.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregar_Click`) es donde se recoge la información del usuario y del producto para mostrar el mensaje de simulación.
- **Explicación:** Al hacer clic en el botón `btnAgregar`, se activa el evento `btnAgregar_Click`. El código obtiene la cadena del elemento seleccionado en `cmbUsuarios` (que representa el tipo de usuario), el texto de `txtProducto` y el texto de `txtCantidad`. Luego, muestra un `MessageBox` que simula la acción, indicando **"Usuario: [Tipo de Usuario] agregó Producto: [Nombre Producto] Cantidad: [Cantidad]"**. Este programa ilustra cómo se podría registrar una acción de venta con el contexto de un usuario específico, sin implicar una lógica de backend o almacenamiento real.

Programa 251. Sistema de ventas con permisos

- **¿En qué consiste el ejercicio?** Este programa demuestra cómo la acción de **"eliminar venta" puede estar restringida a ciertos roles de usuario**, como "Admin", basándose en la selección de un control `ComboBox`.
- **Controles necesarios en el formulario:**
    - **ComboBox:**
        - **Name:** `cmbRoles`
        - **Items:** `"Admin"`, `"Vendedor"` (se asume que se añaden en el diseñador o en `Form_Load`)
        - **Utilización:** Permite al usuario **seleccionar el rol** que está intentando realizar la acción de eliminación.

        - **Propiedades y su uso:** La propiedad `SelectedItem` se utiliza para obtener la cadena del rol seleccionado y compararla con `"Admin"` para determinar si se tienen permisos.
    - **Button:**
        - **Name:** `btnEliminar`
        - **Text:** `"Eliminar Venta"`
        - **Utilización:** El botón que el usuario presiona para **simular la eliminación de una venta**.
        - **Propiedades y su uso:** El evento `Click` (`btnEliminar_Click`) es donde se ejecuta la lógica de verificación de permisos. La propiedad `Text` etiqueta el botón con su acción.
- **Explicación:** Cuando el usuario hace clic en el botón `btnEliminar`, se activa el evento `btnEliminar_Click`. El código dentro de este evento verifica la cadena del `cmbRoles.SelectedItem`.
    - Si el rol seleccionado es **"Admin"**, se muestra un `MessageBox` con el mensaje **"Venta eliminada"**, simulando que la acción se ha realizado con éxito.
    - Si el rol seleccionado **no es "Admin"** (por ejemplo, "Vendedor"), se muestra un `MessageBox` con el mensaje **"No tiene permisos para eliminar"**, indicando la restricción. Este programa es un ejemplo básico de control de acceso basado en roles.

Programa 252. Sistema de ventas con auditoría

- **¿En qué consiste el ejercicio?** Este programa registra cada adición de ventas en un control `ListBox` específico para **fines de auditoría**, incluyendo la fecha, hora y el nombre del producto vendido.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtProducto`
        - **Utilización:** Se usa para que el usuario ingrese el **nombre del producto** que se ha vendido.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del producto introducido. Después de agregar, `txtProducto.Clear()` se usa para limpiar el campo.
    - **TextBox:**
        - **Name:** `txtCantidad`
        - **Utilización:** Se usa para que el usuario ingrese la **cantidad del producto** vendido.
        - **Propiedades y su uso:** La propiedad `Text` se usa para obtener la cantidad. Después de agregar, `txtCantidad.Clear()` se usa para limpiar el campo.
    - **Button:**
        - **Name:** `btnAgregar`

        - **Text:** `"Agregar Venta"`
        - **Utilización:** Es el botón que el usuario presiona para **añadir el producto y su cantidad** a la lista de ventas y registrar la acción en la auditoría.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregar_Click`) es el disparador principal del programa. Al hacer clic, se ejecuta el código que concatena la información y la añade a ambos `ListBox`. La propiedad `Text` (`"Agregar Venta"`) proporciona una etiqueta clara para la acción del botón.
    - **ListBox:**
        - **Name:** `lstVentas`
        - **Utilización:** Se usa para **mostrar los productos y sus cantidades** que se han agregado a la lista de ventas principal.
        - **Propiedades y su uso:** La colección `Items` del `ListBox` es donde se añade la entrada principal de la venta, con `lstVentas.Items.Add()`.
    - **ListBox:**
        - **Name:** `lstAuditoria`
        - **Utilización:** Se usa para **mostrar los registros de auditoría** de cada adición de venta.
        - **Propiedades y su uso:** La colección `Items` de este `ListBox` se utiliza para añadir una nueva entrada de auditoría formateada con la fecha, hora y el nombre del producto vendido, usando `lstAuditoria.Items.Add()`.
- **Explicación:** Cuando el usuario hace clic en el botón `btnAgregar`, se activa el evento `btnAgregar_Click`. Dentro de este evento, el programa realiza dos acciones principales: primero, añade una cadena combinando el nombre del producto (`txtProducto.Text`) y su cantidad (`txtCantidad.Text`) a `lstVentas`. Segundo, registra una entrada en `lstAuditoria` que incluye la **fecha y hora actuales** (obtenida mediante `DateTime.Now.ToString()`) y una descripción del evento: "Venta agregada: [nombre del producto]". Después de estas operaciones, los campos `txtProducto` y `txtCantidad` se limpian automáticamente para la siguiente entrada.

Programa 253. Sistema de ventas con conexión remota

- **¿En qué consiste el ejercicio?** Este programa es un **ejemplo conceptual que simula la conexión a un servidor remoto** para la gestión de ventas, sin implementar la complejidad real de una conexión de red.
- **Controles necesarios en el formulario:**
    - **Button:**
        - **Name:** `btnConectar`
        - **Text:** `"Conectar a Servidor"`
        - **Utilización:** Es el botón que el usuario presiona para **simular el intento de conexión** a un servidor remoto.

        - **Propiedades y su uso:** El evento `Click` (`btnConectar_Click`) es el único controlador de eventos en este programa, y simplemente muestra un mensaje. La propiedad `Text` etiqueta el botón con su acción.
- **Explicación:** Cuando el usuario hace clic en el botón `btnConectar`, se ejecuta el evento `btnConectar_Click`. El código dentro de este evento es muy simple y consiste en una única línea que muestra un `MessageBox` con el mensaje: **"Conexión remota establecida (simulada)"**. La explicación del programa aclara que este es solo un concepto y que una implementación real de una conexión remota requeriría un código de red mucho más complejo, que no se incluye en este ejemplo.

Programa 254. Sistema de ventas con backup automático

- **¿En qué consiste el ejercicio?** Este programa es un **ejemplo conceptual que simula la realización de un backup automático** de los datos de ventas.
- **Controles necesarios en el formulario:**
    - **Button:**
        - **Name:** `btnBackup`
        - **Text:** `"Realizar Backup"`
        - **Utilización:** Es el botón que el usuario presiona para **simular la ejecución de un backup** de los datos de ventas.
        - **Propiedades y su uso:** El evento `Click` (`btnBackup_Click`) es el único controlador de eventos en este programa, y simplemente muestra un mensaje. La propiedad `Text` etiqueta el botón con su acción.
- **Explicación:** Al hacer clic en el botón `btnBackup`, se ejecuta el evento `btnBackup_Click`. El código dentro de este evento es una única línea que muestra un `MessageBox` con el mensaje: **"Backup automático de ventas realizado correctamente"**. El programa conceptualiza la idea de un backup automático, pero la implementación real implicaría operaciones como copiar archivos de bases de datos, exportar datos o utilizar herramientas específicas de backup, las cuales no están presentes en este ejemplo.

Programa 255. Sistema de ventas con restauración de datos

- **¿En qué consiste el ejercicio?** Este programa es un **ejemplo conceptual que simula la restauración de datos** de ventas a partir de un backup previamente realizado.
- **Controles necesarios en el formulario:**
    - **Button:**
        - **Name:** `btnRestaurar`
        - **Text:** `"Restaurar Backup"`
        - **Utilización:** Es el botón que el usuario presiona para **simular la restauración de los datos** de ventas.

        - **Propiedades y su uso:** El evento `Click` (`btnRestaurar_Click`) es el único controlador de eventos en este programa, y simplemente muestra un mensaje. La propiedad `Text` etiqueta el botón con su acción.
- **Explicación:** Cuando el usuario hace clic en el botón `btnRestaurar`, se ejecuta el evento `btnRestaurar_Click`. El código dentro de este evento es una única línea que muestra un `MessageBox` con el mensaje: **"Ventas restauradas desde backup"**. Este programa ilustra conceptualmente la acción de restaurar datos, pero una implementación real requeriría operaciones más complejas como reemplazar archivos de bases de datos o importar datos de un archivo de backup anterior.

Programa 256. Sistema de ventas con notificaciones

- **¿En qué consiste el ejercicio?** Este programa utiliza un control `NotifyIcon` para **mostrar una notificación emergente en la bandeja del sistema (system tray)**, alertando al usuario sobre una nueva venta registrada.
- **Controles necesarios en el formulario:**
    - **NotifyIcon:**
        - **Name:** `NotifyIcon1`
        - **Icon:** `icono.ico` (se requiere un archivo de ícono)
        - **Utilización:** Es el control que permite a la aplicación mostrar un icono en la bandeja del sistema y **enviar notificaciones tipo "globo"** al usuario.
        - **Propiedades y su uso:** `NotifyIcon1.Visible = True` asegura que el icono de la aplicación esté visible en la bandeja del sistema. `NotifyIcon1.ShowBalloonTip()` es el método principal que muestra la notificación. Sus parámetros controlan la duración (3000 ms = 3 segundos), el título de la notificación ("Ventas"), el mensaje ("Nueva venta registrada") y el tipo de icono que acompaña la notificación (`ToolTipIcon.Info` para indicar información).
    - **Button:**
        - **Name:** `btnNotificar`
        - **Text:** `"Notificar Venta"`
        - **Utilización:** Es el botón que el usuario presiona para **simular y activar la notificación** de una nueva venta.
        - **Propiedades y su uso:** El evento `Click` (`btnNotificar_Click`) es el disparador que ejecuta la lógica para mostrar la notificación. La propiedad `Text` etiqueta el botón con su acción.
- **Explicación:** Al hacer clic en el botón `btnNotificar`, se activa el evento `btnNotificar_Click`. Dentro de este evento, la propiedad `NotifyIcon1.Visible` se establece en `True` para asegurar que el icono de la aplicación aparezca en la bandeja del sistema si aún no lo está. Luego, el método `NotifyIcon1.ShowBalloonTip()` se invoca para mostrar la notificación. Esta notificación tiene un título de "Ventas", un mensaje de "Nueva venta registrada", una duración de 3 segundos y utiliza un icono de información (`ToolTipIcon.Info`) para indicar la naturaleza del mensaje. **Requisito adicional:** Para

que `NotifyIcon1` funcione correctamente, se debe haber agregado un archivo de ícono (por ejemplo, `icono.ico`) a la propiedad `Icon` del control en tiempo de diseño.

Programa 257. Sistema de ventas con control de stock

- **¿En qué consiste el ejercicio?** Este programa simula la **gestión de stock** al intentar realizar una venta. Si el stock es suficiente, la cantidad se decrementa; de lo contrario, se muestra una alerta de stock insuficiente.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtProducto`
        - **Utilización:** Campo para que el usuario ingrese el **nombre del producto** que desea vender.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del producto ingresado por el usuario.
    - **TextBox:**
        - **Name:** `txtCantidad`
        - **Utilización:** Campo para que el usuario ingrese la **cantidad del producto** a vender.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener la cantidad ingresada, la cual se convierte a un valor numérico (`Val()`) para las operaciones de stock.
    - **Button:**
        - **Name:** `btnAgregar`
        - **Text:** `"Agregar Venta"`
        - **Utilización:** Es el botón que el usuario presiona para **intentar registrar una venta** y activar la lógica de control de stock.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregar_Click`) es el disparador. Al hacer clic, se ejecuta el código que verifica y actualiza el stock.
- **Explicación:** Se declara un `Dictionary` llamado `stock` a nivel de clase, que se inicializa con algunos productos y sus cantidades de stock (por ejemplo, "ProductoA" con 10 unidades y "ProductoB" con 5). Cuando el usuario hace clic en el botón `btnAgregar`, se activa el evento `btnAgregar_Click`. El programa obtiene el nombre del producto de `txtProducto.Text` y la cantidad de `txtCantidad.Text` (convertida a entero con `Val()`). Luego, verifica dos condiciones: si el `stock` contiene la clave del `producto` (`stock.ContainsKey(producto)`) Y si la `cantidad` solicitada es menor o igual al stock disponible para ese producto (`stock(producto) >= cantidad`).
    - Si ambas condiciones son verdaderas (stock suficiente), el stock del `producto` se decrementa por la `cantidad` vendida (`stock(producto) -= cantidad`), y se muestra un `MessageBox` con el mensaje **"Venta realizada. Stock restante: [cantidad restante]"**.

- Si el stock no es suficiente, se muestra un `MessageBox` con el mensaje **"Stock insuficiente"**.

Programa 258. Sistema de ventas con caducidades

- **¿En qué consiste el ejercicio?** Este programa permite al usuario **registrar ventas incluyendo la fecha de caducidad del producto vendido**, la cual se selecciona mediante un control `DateTimePicker`.
- **Controles necesarios en el formulario:**
  - **TextBox:**
    - **Name:** `txtProducto`
    - **Utilización:** Se usa para que el usuario ingrese el **nombre del producto** vendido.
    - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del producto. Después de agregar, `txtProducto.Clear()` se usa para limpiar el campo.
  - **DateTimePicker:**
    - **Name:** `dtpCaducidad`
    - **Utilización:** Permite al usuario **seleccionar una fecha de caducidad** para el producto vendido.
    - **Propiedades y su uso:** La propiedad `Value` obtiene la fecha y hora seleccionadas. `ToShortDateString()` se utiliza para formatear esta fecha a una cadena corta y legible (ej. "DD/MM/AAAA").
  - **TextBox:**
    - **Name:** `txtCantidad`
    - **Utilización:** Se usa para que el usuario ingrese la **cantidad del producto** vendido.
    - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener la cantidad. Después de agregar, `txtCantidad.Clear()` se usa para limpiar el campo.
  - **Button:**
    - **Name:** `btnAgregar`
    - **Text:** `"Agregar Venta"`
    - **Utilización:** Es el botón que el usuario presiona para **añadir el producto, su cantidad y la fecha de caducidad** a la lista de ventas.
    - **Propiedades y su uso:** El evento `Click` (`btnAgregar_Click`) es el disparador. Al hacer clic, se ejecuta el código que concatena la información y la añade al `ListBox`.
  - **ListBox:**
    - **Name:** `lstVentas`
    - **Utilización:** Se usa para **mostrar los productos vendidos**, incluyendo su nombre, cantidad y fecha de caducidad.

        - **Propiedades y su uso:** `lstVentas.Items.Add()` se utiliza para añadir una cadena formateada con "Producto: [Nombre] - Cantidad: [Cantidad] - Caduca: [Fecha]" a la lista.
- **Explicación:** Cuando el usuario hace clic en el botón `btnAgregar`, se activa el evento `btnAgregar_Click`. El programa obtiene el nombre del producto de `txtProducto.Text`, la cantidad de `txtCantidad.Text`, y la fecha seleccionada en `dtpCaducidad` (formateada como cadena corta con `dtpCaducidad.Value.ToShortDateString()`). Toda esta información se combina en una cadena de texto descriptiva (por ejemplo, "Producto: Leche - Cantidad: 5 - Caduca: 20/03/2025"). Esta cadena se añade como un nuevo elemento a la colección `Items` del `lstVentas`. Finalmente, los campos `txtProducto` y `txtCantidad` se limpian automáticamente para la siguiente entrada.

Programa 259. Sistema de ventas con código de barras

- **¿En qué consiste el ejercicio?** Este programa permite al usuario **ingresar un "código de barras" (tratado como texto simple)** y registrarlo como una venta en una lista.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtCodigo`
        - **Utilización:** Se usa para que el usuario ingrese el **código de barras** del producto vendido (como texto).
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener la entrada del usuario. Después de agregar, `txtCodigo.Clear()` limpia el campo para la siguiente entrada.
    - **Button:**
        - **Name:** `btnAgregar`
        - **Text:** `"Agregar Venta"`
        - **Utilización:** Es el botón que el usuario presiona para **añadir el código ingresado** a la lista de ventas.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregar_Click`) se encarga de añadir el texto del `TextBox` al `ListBox`.
    - **ListBox:**
        - **Name:** `lstVentas`
        - **Utilización:** Se usa para **mostrar la lista de códigos de barra** de las ventas registradas.
        - **Propiedades y su uso:** `lstVentas.Items.Add()` se utiliza para añadir el texto directamente desde `txtCodigo.Text` como un nuevo elemento a la lista, precedido por "Código: ".
- **Explicación:** Cuando el usuario hace clic en el botón `btnAgregar`, se activa el evento `btnAgregar_Click`. El programa toma el texto ingresado en `txtCodigo.Text` y lo añade directamente como un nuevo elemento a la colección `Items` del `lstVentas`, formateado como "Código: [código de barras]". Inmediatamente después, el campo

`txtCodigo` se vacía utilizando `txtCodigo.Clear()` para que el usuario pueda ingresar otro código.

Programa 260. Sistema de ventas con descuentos

- **¿En qué consiste el ejercicio?** Este programa calcula el **precio final de una venta aplicando un descuento porcentual** y añade esta información, junto con el producto, a una lista de ventas.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtProducto`
        - **Utilización:** Campo para el **nombre del producto** vendido.
        - **Propiedades y su uso:** La propiedad `Text` contiene el nombre del producto. Se limpia con `txtProducto.Clear()` después de añadir la venta.
    - **TextBox:**
        - **Name:** `txtPrecio`
        - **Utilización:** Campo para el **precio original** del producto.
        - **Propiedades y su uso:** `Text` obtiene el precio, y `Val()` lo convierte a un valor numérico para los cálculos. Se limpia con `txtPrecio.Clear()`.
    - **TextBox:**
        - **Name:** `txtDescuento`
        - **Utilización:** Campo para el **porcentaje de descuento** a aplicar.
        - **Propiedades y su uso:** `Text` obtiene el descuento, y `Val()` lo convierte a un valor numérico. Se limpia con `txtDescuento.Clear()`.
    - **Button:**
        - **Name:** `btnAgregar`
        - **Text:** `"Agregar Venta"`
        - **Utilización:** Botón para **añadir la venta con el descuento calculado** a la lista.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregar_Click`) realiza los cálculos del descuento y la adición a la lista.
    - **ListBox:**
        - **Name:** `lstVentas`
        - **Utilización:** Muestra los **detalles de la venta**, incluyendo producto, precio original, porcentaje de descuento y el total final.
        - **Propiedades y su uso:** `lstVentas.Items.Add()` añade una cadena formateada con "Producto: [Nombre] - Precio: [Original] - Descuento: [X]% - Total: [Final]".
- **Explicación:** Cuando el usuario hace clic en el botón `btnAgregar`, se activa el evento `btnAgregar_Click`. Primero, los valores de `txtPrecio.Text` y `txtDescuento.Text` se convierten a números usando `Val()`. Luego, se calcula el `total` aplicando el descuento con la fórmula: `precio - (precio * descuento / 100)`. Finalmente, se forma una cadena de texto combinando el nombre del producto

(`txtProducto.Text`), el `precio` original, el `descuento` porcentual y el `total` calculado. Esta cadena se añade como un nuevo elemento a la colección `Items` del `lstVentas`. Después de añadir el elemento, los tres `TextBox` (`txtProducto`, `txtPrecio`, `txtDescuento`) se limpian automáticamente para facilitar la entrada de nuevas ventas.

Programa 261. Sistema escolar básico

- **¿En qué consiste el ejercicio?** Este programa implementa un **sistema escolar básico que permite registrar alumnos y materias** en un control `ListBox` para su visualización.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtAlumno`
        - **Utilización:** Se usa para que el usuario ingrese el **nombre del alumno** que desea registrar.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del alumno introducido. Después de agregar el registro, `txtAlumno.Clear()` se usa para limpiar el campo.
    - **TextBox:**
        - **Name:** `txtMateria`
        - **Utilización:** Se usa para que el usuario ingrese el **nombre de la materia** a registrar.
        - **Propiedades y su uso:** La propiedad `Text` se usa para obtener el nombre de la materia. Después de agregar el registro, `txtMateria.Clear()` se usa para limpiar el campo.
    - **Button:**
        - **Name:** `btnAgregar`
        - **Text:** `"Agregar Registro"`
        - **Utilización:** Es el botón que el usuario presiona para **añadir el alumno y la materia** a la lista de registros.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregar_Click`) se activa al hacer clic en el botón. Este evento es el disparador para ejecutar el código que concatena la información y la añade al `ListBox`. La propiedad `Text` (`"Agregar Registro"`) proporciona una etiqueta clara para la acción del botón.
    - **ListBox:**
        - **Name:** `lstRegistros`
        - **Utilización:** Se usa para **mostrar los registros de alumnos y materias** que se han agregado al sistema.
        - **Propiedades y su uso:** La colección `Items` del `ListBox` es donde se añade la entrada del registro. `lstRegistros.Items.Add()` se utiliza para añadir una nueva entrada de texto, combinando el nombre del alumno y la materia, por ejemplo, "Alumno: X - Materia: Y".

- **Explicación:** Cuando el usuario hace clic en el botón `btnAgregar`, se activa el evento `btnAgregar_Click`. Dentro de este evento, el programa toma el texto de `txtAlumno` y `txtMateria`, los concatena en una cadena de texto descriptiva (como "Alumno: X - Materia: Y"), y luego añade esta cadena como un nuevo elemento a la colección `Items` del `lstRegistros`. Después de añadir el elemento, los campos `txtAlumno` y `txtMateria` se limpian automáticamente para facilitar la entrada de nuevos registros.

Programa 262. Sistema escolar con base de datos

- **¿En qué consiste el ejercicio?** Este programa utiliza un **`DataTable` como una simulación de una base de datos en memoria** para almacenar registros de alumnos y materias, y los muestra en un control `DataGridView`.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtAlumno`
        - **Utilización:** Se usa para que el usuario introduzca el **nombre del alumno** a registrar.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del alumno. Después de añadir el registro, `txtAlumno.Clear()` limpia el campo.
    - **TextBox:**
        - **Name:** `txtMateria`
        - **Utilización:** Se usa para que el usuario introduzca la **materia** a registrar.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener la materia. Después de añadir el registro, `txtMateria.Clear()` limpia el campo.
    - **Button:**
        - **Name:** `btnAgregar`
        - **Text:** `"Agregar Registro"`
        - **Utilización:** Es el botón para **añadir el alumno y la materia** al `DataTable` y, por ende, al `DataGridView`.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregar_Click`) se encarga de crear una nueva fila en el `DataTable` con los datos de los `TextBox`.
    - **DataGridView:**
        - **Name:** `dgvRegistros`
        - **Utilización:** Se usa para **visualizar los registros de alumnos y materias** en un formato tabular, enlazado directamente a los datos del `DataTable`.
        - **Propiedades y su uso:** La propiedad `DataSource` es clave; `dgvRegistros.DataSource = dt` vincula el control al `DataTable` `dt`, haciendo que cualquier cambio en `dt` se refleje automáticamente en el `DataGridView`. Las columnas "Alumno" y "Materia" se definen programáticamente en el evento `Form1_Load` con `dt.Columns.Add()`.

- **Explicación:** Al inicio del programa (en el evento `Form1_Load`), se declara un objeto `DataTable` llamado `dt` a nivel de clase. Dentro de `Form1_Load`, se le definen dos columnas: "Alumno" y "Materia". Luego, el `dgvRegistros` se enlaza a este `DataTable` mediante su propiedad `DataSource`. Cuando el usuario hace clic en `btnAgregar`, el evento `btnAgregar_Click` añade una nueva fila a `dt.Rows` utilizando el texto de `txtAlumno` y `txtMateria`. Debido al enlace de datos (`DataSource`), el `DataGridView` se actualiza automáticamente para mostrar la nueva entrada. Los `TextBox` se limpian para la siguiente entrada.

Programa 263. Sistema escolar con registro de alumnos

- **¿En qué consiste el ejercicio?** Este programa permite al usuario **registrar nuevos alumnos con su nombre y edad** en una lista.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtNombreAlumno`
        - **Utilización:** Campo de texto donde el usuario introduce el **nombre del alumno**.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del alumno. Después de agregar, `txtNombreAlumno.Clear()` limpia el campo.
    - **TextBox:**
        - **Name:** `txtEdad`
        - **Utilización:** Campo de texto donde el usuario introduce la **edad del alumno**.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener la edad. Después de agregar, `txtEdad.Clear()` limpia el campo.
    - **Button:**
        - **Name:** `btnAgregarAlumno`
        - **Text:** `"Agregar Alumno"`
        - **Utilización:** El botón que el usuario presiona para **añadir la información del alumno** a la lista.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregarAlumno_Click`) es el que contiene la lógica para combinar y añadir los datos a la lista.
    - **ListBox:**
        - **Name:** `lstAlumnos`
        - **Utilización:** Muestra los **alumnos registrados**, incluyendo su nombre y edad.
        - **Propiedades y su uso:** `lstAlumnos.Items.Add()` se utiliza para añadir una cadena formateada con "Nombre: X - Edad: Y" a la lista.
- **Explicación:** Cuando el usuario hace clic en el botón `btnAgregarAlumno`, se activa el evento `btnAgregarAlumno_Click`. El código toma el texto de `txtNombreAlumno` y `txtEdad`, los combina en una cadena de texto descriptiva (como "Nombre: [Nombre del Alumno] - Edad: [Edad del Alumno]"), y esta cadena se añade como un nuevo elemento a

`lstAlumnos.Items`. Finalmente, los campos `txtNombreAlumno` y `txtEdad` se limpian para la siguiente entrada.

Programa 264. Sistema escolar con registro de profesores

- **¿En qué consiste el ejercicio?** Este programa permite **registrar nuevos profesores con su nombre y la materia que imparten** en una lista.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtNombreProfesor`
        - **Utilización:** Campo de texto donde el usuario introduce el **nombre del profesor**.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del profesor. Después de agregar, `txtNombreProfesor.Clear()` limpia el campo.
    - **TextBox:**
        - **Name:** `txtMateria`
        - **Utilización:** Campo de texto donde el usuario introduce la **materia que imparte el profesor**.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener la materia. Después de agregar, `txtMateria.Clear()` limpia el campo.
    - **Button:**
        - **Name:** `btnAgregarProfesor`
        - **Text:** `"Agregar Profesor"`
        - **Utilización:** El botón que el usuario presiona para **añadir la información del profesor** a la lista.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregarProfesor_Click`) es el que contiene la lógica para combinar y añadir los datos a la lista.
    - **ListBox:**
        - **Name:** `lstProfesores`
        - **Utilización:** Muestra los **profesores registrados**, incluyendo su nombre y la materia que imparten.
        - **Propiedades y su uso:** `lstProfesores.Items.Add()` se utiliza para añadir una cadena formateada con "Profesor: X - Materia: Y" a la lista.
- **Explicación:** Cuando el usuario hace clic en el botón `btnAgregarProfesor`, se activa el evento `btnAgregarProfesor_Click`. El código toma el texto de `txtNombreProfesor` y `txtMateria`, los combina en una cadena de texto descriptiva (como "Profesor: [Nombre del Profesor] - Materia: [Nombre de la Materia]"), y esta cadena se añade como un nuevo elemento a `lstProfesores.Items`. Finalmente, los campos `txtNombreProfesor` y `txtMateria` se limpian para la siguiente entrada.

Programa 265. Sistema escolar con registro de materias

- **¿En qué consiste el ejercicio?** Este programa permite al usuario **agregar nuevas materias** a una lista.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtMateria`
        - **Utilización:** Campo de texto donde el usuario introduce el **nombre de la materia** a agregar.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre de la materia. Después de agregar, `txtMateria.Clear()` limpia el campo.
    - **Button:**
        - **Name:** `btnAgregarMateria`
        - **Text:** `"Agregar Materia"`
        - **Utilización:** El botón que el usuario presiona para **añadir la materia** a la lista.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregarMateria_Click`) es el que contiene la lógica para añadir la materia al `ListBox`.
    - **ListBox:**
        - **Name:** `lstMaterias`
        - **Utilización:** Muestra las **materias registradas**.
        - **Propiedades y su uso:** `lstMaterias.Items.Add()` se utiliza para añadir el texto directamente desde `txtMateria.Text` como un nuevo elemento a la lista.
- **Explicación:** Cuando el usuario hace clic en el botón `btnAgregarMateria`, se activa el evento `btnAgregarMateria_Click`. El programa toma el texto ingresado en `txtMateria.Text` y lo añade directamente como un nuevo elemento a la colección `Items` del `lstMaterias`. Inmediatamente después, el campo `txtMateria` se vacía utilizando `txtMateria.Clear()` para que el usuario pueda ingresar otra materia.

Programa 266. Sistema escolar con registro de calificaciones

- **¿En qué consiste el ejercicio?** Este programa permite **registrar calificaciones de alumnos para una materia específica**.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtAlumno`
        - **Utilización:** Campo para que el usuario ingrese el **nombre del alumno** cuya calificación se va a registrar.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del alumno. Después de agregar, `txtAlumno.Clear()` limpia el campo.
    - **TextBox:**

        - **Name:** `txtMateria`
        - **Utilización:** Campo para que el usuario ingrese el **nombre de la materia**.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener la materia. Después de agregar, `txtMateria.Clear()` limpia el campo.
    - **TextBox:**
        - **Name:** `txtCalificacion`
        - **Utilización:** Campo para que el usuario ingrese la **nota o calificación**.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener la calificación. Después de agregar, `txtCalificacion.Clear()` limpia el campo.
    - **Button:**
        - **Name:** `btnAgregarCalificacion`
        - **Text:** `"Agregar Calificación"`
        - **Utilización:** Es el botón que el usuario presiona para **añadir la calificación** al sistema.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregarCalificacion_Click`) es el disparador. Al hacer clic, se ejecuta el código que concatena la información y la añade al `ListBox`.
    - **ListBox:**
        - **Name:** `lstCalificaciones`
        - **Utilización:** Se usa para **mostrar las calificaciones registradas**, incluyendo el alumno, la materia y la nota.
        - **Propiedades y su uso:** `lstCalificaciones.Items.Add()` se utiliza para añadir una cadena formateada con "Alumno: X - Materia: Y - Nota: Z" a la lista.
- **Explicación:** Cuando el usuario hace clic en el botón `btnAgregarCalificacion`, se activa el evento `btnAgregarCalificacion_Click`. El programa toma el texto de `txtAlumno`, `txtMateria` y `txtCalificacion`, los combina en una cadena de texto descriptiva (por ejemplo, "Alumno: [Nombre del Alumno] - Materia: [Nombre de la Materia] - Nota: [Calificación]"), y luego añade esta cadena como un nuevo elemento a la colección `Items` del `lstCalificaciones`. Finalmente, los tres `TextBox` (`txtAlumno`, `txtMateria`, y `txtCalificacion`) se limpian automáticamente para facilitar la entrada de nuevas calificaciones.

Programa 267. Sistema escolar con reportes

- **¿En qué consiste el ejercicio?** Este programa genera un **reporte simple de las calificaciones** al recopilar todos los elementos actualmente listados en un `ListBox` y mostrarlos en un cuadro de mensaje (`MessageBox`).
- **Controles necesarios en el formulario:**
    - **Button:**
        - **Name:** `btnReporte`
        - **Text:** `"Generar Reporte"`

        - **Utilización:** Es el botón que el usuario presiona para **crear y visualizar el reporte** de las calificaciones.
        - **Propiedades y su uso:** El evento `Click` (`btnReporte_Click`) desencadena la recopilación de los datos y la construcción del mensaje del reporte. La propiedad `Text` etiqueta el botón.
    - **ListBox:**
        - **Name:** `lstCalificaciones`
        - **Utilización:** Contiene la **lista de calificaciones** que conformarán el reporte.
        - **Propiedades y su uso:** La colección `Items` se itera (`For Each item In lstCalificaciones.Items`) para obtener cada elemento de calificación. `item.ToString()` convierte cada elemento a su representación de cadena para ser incluido en el reporte.
- **Explicación:** Al hacer clic en el botón `btnReporte`, se ejecuta el evento `btnReporte_Click`. El código inicializa una cadena `reporte` con un título "Reporte de Calificaciones:" y un salto de línea (`Environment.NewLine`). Luego, un bucle `For Each` recorre cada `item` en `lstCalificaciones.Items`. En cada iteración, el `item` se convierte a cadena y se concatena al `reporte`, seguido de otro salto de línea. Una vez que el bucle ha procesado todos los elementos, el `reporte` completo se muestra al usuario en un `MessageBox`.

Programa 268. Sistema escolar con gráficos

- **¿En qué consiste el ejercicio?** Este programa visualiza las calificaciones (o la existencia de registros de calificaciones) que se encuentran en el `lstCalificaciones` mediante un **gráfico de columnas** que se muestra en un control `Chart`.
- **Controles necesarios en el formulario:**
    - **Chart:**
        - **Name:** `chartCalificaciones`
        - **ChartType:** `Column` (configurado en tiempo de ejecución)
        - **Utilización:** Control gráfico que **muestra visualmente los datos** de calificaciones.
        - **Propiedades y su uso:** La propiedad `Series` es una colección donde se añaden los conjuntos de datos. `chartCalificaciones.Series.Clear()` limpia series anteriores. `ChartType` de una serie se establece a `DataVisualization.Charting.SeriesChartType.Column` para que se muestre como un gráfico de columnas. `Points.AddXY(item.ToString(), 1)` añade puntos de datos, usando el nombre de la calificación como etiqueta X y un valor fijo de 1 para representar su existencia.
    - **Button:**
        - **Name:** `btnGraficar`
        - **Text:** `"Mostrar Gráfico"`

        - **Utilización:** Botón para **generar y actualizar el gráfico** con los datos actuales del `lstCalificaciones`.
        - **Propiedades y su uso:** El evento `Click` (`btnGraficar_Click`) es el que contiene la lógica para construir y mostrar el gráfico.
    - **ListBox:**
        - **Name:** `lstCalificaciones`
        - **Utilización:** Contiene la **lista de calificaciones** que se utilizarán como datos para el gráfico.
        - **Propiedades y su uso:** La colección `Items` de `lstCalificaciones` se itera (`For Each item In lstCalificaciones.Items`) para obtener cada elemento de calificación y añadirlo al gráfico.
- **Explicación:** Al hacer clic en el botón `btnGraficar`, se activa el evento `btnGraficar_Click`. El código comienza por **limpiar cualquier serie de datos existente** en `chartCalificaciones` con `chartCalificaciones.Series.Clear()`. Luego, se crea una nueva instancia de `Series` y se nombra "Calificaciones". Se establece el `ChartType` de esta serie como `Column`. A continuación, un bucle `For Each` recorre cada `item` en la colección `lstCalificaciones.Items`. Por cada `item`, se añade un punto al gráfico utilizando `serie.Points.AddXY(item.ToString(), 1)`. Esto significa que cada calificación del `ListBox` aparecerá como una columna en el gráfico, con el nombre de la calificación como etiqueta y una altura de 1 (simbolizando su existencia). Finalmente, esta `serie` se añade a la colección `chartCalificaciones.Series`, lo que hace que el gráfico se dibuje y se muestre en el formulario. **Requisito adicional:** Para utilizar el control `Chart`, es necesario que la referencia `System.Windows.Forms.DataVisualization` esté añadida al proyecto.

Programa 269. Sistema escolar con exportación Excel

- **¿En qué consiste el ejercicio?** Este programa es un **ejemplo conceptual que simula la exportación de datos de calificaciones a una hoja de cálculo de Microsoft Excel**. No contiene una implementación real de la exportación, sino que indica lo que se necesitaría.
- **Controles necesarios en el formulario:**
    - **Button:**
        - **Name:** `btnExportar`
        - **Text:** `"Exportar a Excel"`
        - **Utilización:** Botón que el usuario presiona para **iniciar el proceso conceptual de exportación**.
        - **Propiedades y su uso:** El evento `Click` (`btnExportar_Click`) es el único controlador de eventos en este programa, y simplemente muestra un mensaje. La propiedad `Text` etiqueta el botón.
- **Explicación:** Cuando el usuario hace clic en el botón `btnExportar`, se ejecuta el evento `btnExportar_Click`. El código dentro de este evento es muy simple y consiste en una única línea que muestra un `MessageBox` con el mensaje: **"Calificaciones exportadas a**

Excel (requiere referencia a Microsoft Excel)". **Requisito adicional:** La fuente aclara que, para una implementación real de esta funcionalidad, se necesitaría agregar una referencia a la librería de objetos de Microsoft Excel en el proyecto. Este programa se limita a ilustrar el concepto sin el código complejo de interacción con Excel.

Programa 270. Sistema escolar con importación Excel

- **¿En qué consiste el ejercicio?** Este programa es un **ejemplo conceptual que simula la importación de datos de calificaciones desde una hoja de cálculo de Microsoft Excel**. Al igual que el de exportación, se centra en la idea más que en la implementación completa.
- **Controles necesarios en el formulario:**
    - **Button:**
        - **Name:** `btnImportar`
        - **Text:** `"Importar desde Excel"`
        - **Utilización:** Botón que el usuario presiona para **iniciar el proceso conceptual de importación**.
        - **Propiedades y su uso:** El evento `Click` (`btnImportar_Click`) es el único controlador de eventos y solo muestra un mensaje informativo. La propiedad `Text` etiqueta el botón.
- **Explicación:** Al hacer clic en el botón `btnImportar`, se activa el evento `btnImportar_Click`. El código consiste en una línea que muestra un `MessageBox` con el mensaje: **"Calificaciones importadas desde Excel (requiere referencia a Microsoft Excel)"**. **Requisito adicional:** Similar al programa de exportación, la fuente indica que para una implementación funcional, se requeriría agregar una referencia a la librería de objetos de Microsoft Excel en el proyecto. El programa, por lo tanto, es un marcador de posición para una funcionalidad más compleja.

Programa 271. Sistema escolar con múltiples usuarios

- **¿En qué consiste el ejercicio?** Este programa simula un proceso de inicio de sesión para **diferentes tipos de usuarios (Admin, Profesor, Alumno)** dentro de un sistema escolar, donde el tipo de usuario se selecciona a través de un control `ComboBox`. Este es un ejemplo conceptual que no incluye una autenticación real de credenciales.
- **Controles necesarios en el formulario:**
    - **ComboBox:**
        - **Name:** `cmbUsuarios`
        - **Items:** `"Admin"`, `"Profesor"`, `"Alumno"`
        - **Utilización:** Permite al usuario **seleccionar su rol o tipo de usuario** antes de "acceder" al sistema.
        - **Propiedades y su uso:** La propiedad `SelectedItem` del `ComboBox` se utiliza para obtener la cadena de texto que representa el tipo de usuario

elegido por el usuario (`"Admin"`, `"Profesor"` o `"Alumno"`). Esta selección se usa para el mensaje conceptual de inicio de sesión.

- **Button:**
  - **Name:** `btnAcceder`
  - **Text:** `"Acceder"`
  - **Utilización:** Es el botón que el usuario presiona para **simular el acceso al sistema** con el rol seleccionado.
  - **Propiedades y su uso:** El evento `Click` (`btnAcceder_Click`) es el disparador. Al hacer clic, se muestra un `MessageBox` que indica el tipo de usuario que ha "iniciado sesión". La propiedad `Text` (`"Acceder"`) proporciona una etiqueta clara para la acción.

- **Explicación:** Cuando el usuario hace clic en el botón `btnAcceder`, se activa el evento `btnAcceder_Click`. El código dentro de este evento recupera el tipo de usuario seleccionado del `cmbUsuarios.SelectedItem`. Luego, muestra un `MessageBox` que indica conceptualmente que el "Usuario: [Tipo de Usuario] ha iniciado sesión". Este programa es una demostración conceptual y no realiza ninguna validación de credenciales real ni autenticación contra un sistema de usuarios.

Programa 272. Sistema escolar con permisos

- **¿En qué consiste el ejercicio?** Este programa demuestra cómo la acción de **"eliminar registro" puede estar restringida a ciertos roles de usuario**, como "Admin", basándose en la selección de un control `ComboBox`. Esto es un ejemplo de control de acceso basado en roles.

- **Controles necesarios en el formulario:**

  - **ComboBox:**
    - **Name:** `cmbRoles`
    - **Items:** `"Admin"`, `"Profesor"` (se asume que se añaden en el diseñador o en un evento como `Form_Load`)
    - **Utilización:** Permite al usuario **seleccionar el rol o nivel de permiso** con el que desea realizar la acción de eliminación.
    - **Propiedades y su uso:** La propiedad `SelectedItem` se utiliza para obtener la cadena que representa el rol elegido por el usuario (`"Admin"` o `"Profesor"`). Este valor es crucial para la lógica de verificación de permisos.
  - **Button:**
    - **Name:** `btnEliminarRegistro`
    - **Text:** `"Eliminar Registro"`
    - **Utilización:** El botón que el usuario presiona para **intentar ejecutar la acción de eliminar un registro**.
    - **Propiedades y su uso:** El evento `Click` (`btnEliminarRegistro_Click`) es donde se ejecuta la lógica de

verificación de permisos. La propiedad `Text` (`"Eliminar Registro"`) proporciona una etiqueta clara para la acción.

- **Explicación:** Cuando el usuario hace clic en el botón `btnEliminarRegistro`, se activa el evento `btnEliminarRegistro_Click`. El código dentro de este evento **verifica el rol seleccionado en el `cmbRoles.SelectedItem`**.
    - Si el rol seleccionado es **"Admin"**, se muestra un `MessageBox` con el mensaje **"Registro eliminado"**, simulando que la acción se ha realizado con éxito debido a los permisos adecuados.
    - Si el rol seleccionado **no es "Admin"** (por ejemplo, si se selecciona "Profesor"), se muestra un `MessageBox` indicando **"No tiene permisos para eliminar"**, denegando la acción.

Programa 273. Sistema escolar con auditoría

- **¿En qué consiste el ejercicio?** Este programa implementa un mecanismo de auditoría al **registrar cada adición de un alumno en un control `ListBox` específico**, incluyendo la fecha y hora exactas del evento, así como el nombre del alumno agregado.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtAlumno`
        - **Utilización:** Se usa para que el usuario ingrese el **nombre del alumno** que desea registrar en el sistema escolar.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del alumno introducido por el usuario. Después de registrar la acción, `txtAlumno.Clear()` se usa para limpiar el campo, preparándolo para la siguiente entrada.
    - **Button:**
        - **Name:** `btnAgregarAlumno`
        - **Text:** `"Agregar Alumno"`
        - **Utilización:** Es el botón que el usuario presiona para **añadir un nuevo alumno** y, consecuentemente, generar una entrada en el registro de auditoría.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregarAlumno_Click`) es el disparador principal del programa. Al hacer clic, se ejecuta el código que registra la información de auditoría. La propiedad `Text` (`"Agregar Alumno"`) proporciona una etiqueta clara para la acción del botón.
    - **ListBox:**
        - **Name:** `lstAuditoria`
        - **Utilización:** Se usa para **mostrar los registros de auditoría**, listando cada vez que un alumno es agregado.
        - **Propiedades y su uso:** La colección `Items` de este `ListBox` es fundamental. `lstAuditoria.Items.Add()` se utiliza para añadir una

nueva entrada de texto formateada con la fecha, hora y el nombre del alumno que fue agregado.

- **Explicación:** Cuando el usuario hace clic en el botón `btnAgregarAlumno`, se activa el evento `btnAgregarAlumno_Click`. El código dentro de este evento construye una cadena de auditoría que incluye la **fecha y hora actuales** (obtenida mediante `DateTime.Now.ToString()`) y una descripción del evento: " - Alumno agregado: [nombre del alumno]". Esta cadena se añade como un nuevo elemento a la colección `Items` del `lstAuditoria`, haciendo visible el registro. Finalmente, el campo `txtAlumno` se limpia automáticamente para la siguiente entrada.

Programa 274. Sistema escolar con conexión remota

- **¿En qué consiste el ejercicio?** Este programa es un **ejemplo conceptual que simula la conexión a un servidor remoto** para la gestión del sistema escolar, sin implementar la complejidad real de una conexión de red.
- **Controles necesarios en el formulario:**
    - **Button:**
        - **Name:** `btnConectar`
        - **Text:** `"Conectar a Servidor"`
        - **Utilización:** Es el botón que el usuario presiona para **simular el intento de conexión** a un servidor remoto.
        - **Propiedades y su uso:** El evento `Click` (`btnConectar_Click`) es el único controlador de eventos en este programa. Al ser clicado, simplemente muestra un mensaje informativo. La propiedad `Text` (`"Conectar a Servidor"`) etiqueta el botón con su acción.
- **Explicación:** Al hacer clic en el botón `btnConectar`, se ejecuta el evento `btnConectar_Click`. El código dentro de este evento es muy simple y consiste en una única línea que muestra un `MessageBox` con el mensaje: **"Conexión remota establecida (simulada)"**. La explicación del programa aclara que este es meramente un concepto y que una implementación real de una conexión remota en un sistema escolar requeriría un código de red mucho más complejo y una infraestructura de servidor, que no están incluidos en este ejemplo.

Programa 275. Sistema escolar con backup automático

- **¿En qué consiste el ejercicio?** Este programa es un **ejemplo conceptual que simula la realización de un backup automático** de los datos del sistema escolar. No realiza un backup real, sino que conceptualiza la idea.
- **Controles necesarios en el formulario:**
    - **Button:**
        - **Name:** `btnBackup`
        - **Text:** `"Realizar Backup"`

        - **Utilización:** Es el botón que el usuario presiona para **simular la ejecución de un backup** de los datos escolares.
        - **Propiedades y su uso:** El evento `Click` (`btnBackup_Click`) es el único controlador de eventos en este programa. Al ser clicado, simplemente muestra un mensaje. La propiedad `Text` (`"Realizar Backup"`) etiqueta el botón con su acción.
- **Explicación:** Al hacer clic en el botón `btnBackup`, se ejecuta el evento `btnBackup_Click`. El código dentro de este evento es una única línea que muestra un `MessageBox` con el mensaje: **"Backup automático de datos escolares realizado correctamente"**. Este programa conceptualiza la idea de un backup automático para un sistema escolar, pero la implementación real implicaría operaciones más complejas como copiar archivos de bases de datos, exportar datos a formatos específicos o interactuar con herramientas de backup, las cuales no están presentes en este ejemplo.

Programa 276. Sistema escolar con restauración de datos

- **¿En qué consiste el ejercicio?** Este programa es un **ejemplo conceptual que simula la restauración de datos** del sistema escolar a partir de un backup previamente realizado. No implementa una restauración real, sino que ilustra la idea.
- **Controles necesarios en el formulario:**
    - **Button:**
        - **Name:** `btnRestaurar`
        - **Text:** `"Restaurar Backup"`
        - **Utilización:** Es el botón que el usuario presiona para **simular la restauración de los datos** escolares.
        - **Propiedades y su uso:** El evento `Click` (`btnRestaurar_Click`) es el único controlador de eventos en este programa. Al ser clicado, simplemente muestra un mensaje. La propiedad `Text` (`"Restaurar Backup"`) etiqueta el botón con su acción.
- **Explicación:** Cuando el usuario hace clic en el botón `btnRestaurar`, se ejecuta el evento `btnRestaurar_Click`. El código dentro de este evento es una única línea que muestra un `MessageBox` con el mensaje: **"Datos escolares restaurados desde backup"**. Este programa ilustra conceptualmente la acción de restaurar datos, pero una implementación real requeriría operaciones más complejas como reemplazar archivos de bases de datos, importar datos de un archivo de backup anterior o interactuar con un sistema de gestión de backups.

Programa 277. Sistema escolar con notificaciones

- **¿En qué consiste el ejercicio?** Este programa utiliza un control `NotifyIcon` para **mostrar una notificación emergente en la bandeja del sistema (system tray)**, alertando al usuario sobre un "Nuevo evento registrado" en el sistema escolar.
- **Controles necesarios en el formulario:**

    - **NotifyIcon:**
        - **Name:** `NotifyIcon1`
        - **Icon:** `icono.ico` (se requiere un archivo de ícono para que el icono de la aplicación sea visible en la bandeja del sistema)
        - **Utilización:** Este control permite a la aplicación mostrar un icono en la bandeja del sistema y **enviar notificaciones tipo "globo"** al usuario.
        - **Propiedades y su uso:** La propiedad `Visible` se establece en `True` (`NotifyIcon1.Visible = True`) para que el icono aparezca en la bandeja del sistema. El método `ShowBalloonTip()` es el principal para mostrar la notificación, con parámetros que controlan la duración (3000 ms = 3 segundos), el título de la notificación ("Sistema Escolar"), el mensaje ("Nuevo evento registrado") y el tipo de icono que acompaña la notificación (`ToolTipIcon.Info` para indicar información).
    - **Button:**
        - **Name:** `btnNotificar`
        - **Text:** `"Notificar Evento"`
        - **Utilización:** Es el botón que el usuario presiona para **simular y activar la notificación** de un nuevo evento.
        - **Propiedades y su uso:** El evento `Click` (`btnNotificar_Click`) es el disparador que ejecuta la lógica para mostrar la notificación. La propiedad `Text` (`"Notificar Evento"`) etiqueta el botón con su acción.
- **Explicación:** Al hacer clic en el botón `btnNotificar`, se activa el evento `btnNotificar_Click`. Dentro de este evento, la propiedad `NotifyIcon1.Visible` se establece en `True` para asegurar que el icono de la aplicación aparezca en la bandeja del sistema. Luego, el método `NotifyIcon1.ShowBalloonTip()` se invoca para mostrar la notificación emergente. Esta notificación tiene un título de "Sistema Escolar", un mensaje de "Nuevo evento registrado", una duración de 3 segundos y utiliza un icono de información (`ToolTipIcon.Info`).

Programa 278. Sistema escolar con estadísticas

- **¿En qué consiste el ejercicio?** Este programa proporciona **estadísticas básicas del sistema escolar** al mostrar el número total de registros (por ejemplo, calificaciones, alumnos, etc., en este contexto representados por los elementos de un `ListBox`) mediante un cuadro de mensaje (`MessageBox`).
- **Controles necesarios en el formulario:**
    - **ListBox:**
        - **Name:** `lstCalificaciones`
        - **Utilización:** Contiene la **lista de registros** del sistema escolar (en este caso, calificaciones) que se utilizarán para calcular las estadísticas. Aunque se nombra `lstCalificaciones`, en el contexto de "total de registros" puede albergar cualquier tipo de entrada para contar.

        - **Propiedades y su uso:** La propiedad `Items.Count` de `lstCalificaciones` es crucial, ya que devuelve el número de elementos que hay en la lista, lo que se interpreta como el total de registros.
    - **Button:**
        - **Name:** `btnEstadisticas`
        - **Text:** `"Mostrar Estadísticas"`
        - **Utilización:** Es el botón que el usuario presiona para **obtener y visualizar las estadísticas** del sistema escolar.
        - **Propiedades y su uso:** El evento `Click` (`btnEstadisticas_Click`) es el disparador. Al hacer clic, se ejecuta el código que calcula y muestra el total de registros. La propiedad `Text` (`"Mostrar Estadísticas"`) etiqueta el botón.
- **Explicación:** Cuando el usuario hace clic en el botón `btnEstadisticas`, se activa el evento `btnEstadisticas_Click`. Dentro de este evento, el programa obtiene el número de elementos presentes en `lstCalificaciones` utilizando `lstCalificaciones.Items.Count` y lo almacena en la variable `totalAlumnos` (que en este contexto representa el total de registros). Finalmente, un `MessageBox` se encarga de mostrar al usuario esta estadística, indicando **"Total de registros: [número]"**.

Programa 279. Sistema escolar con boletines

- **¿En qué consiste el ejercicio?** Este programa genera un **boletín de calificaciones simple** al recopilar todos los elementos actualmente listados en un control `ListBox` (asumiendo que son calificaciones) y mostrarlos en un cuadro de mensaje (`MessageBox`).
- **Controles necesarios en el formulario:**
    - **Button:**
        - **Name:** `btnGenerarBoletin`
        - **Text:** `"Generar Boletín"`
        - **Utilización:** Es el botón que el usuario presiona para **crear y visualizar el boletín** de calificaciones.
        - **Propiedades y su uso:** El evento `Click` (`btnGenerarBoletin_Click`) desencadena la recopilación de los datos y la construcción del mensaje del boletín. La propiedad `Text` etiqueta el botón con su acción.
    - **ListBox:**
        - **Name:** `lstCalificaciones`
        - **Utilización:** Contiene la **lista de calificaciones** que conformarán el boletín.
        - **Propiedades y su uso:** La colección `Items` se itera (`For Each item In lstCalificaciones.Items`) para obtener cada elemento de calificación. `item.ToString()` convierte cada elemento a su representación de cadena para ser incluido en el boletín.
- **Explicación:** Al hacer clic en el botón `btnGenerarBoletin`, se ejecuta el evento `btnGenerarBoletin_Click`. El código inicializa una cadena `boletin` con un título "Boletín de Calificaciones:" y un salto de línea (`Environment.NewLine`). Luego, un

bucle `For Each` recorre cada `item` en `lstCalificaciones.Items`. En cada iteración, el `item` se convierte a cadena y se concatena al `boletin`, seguido de otro salto de línea. Una vez que el bucle ha procesado todos los elementos, el `boletin` completo se muestra al usuario en un `MessageBox`.

Programa 280. Sistema escolar con login de usuarios

- **¿En qué consiste el ejercicio?** Este programa implementa un **sistema de login básico para el sistema escolar con un nombre de usuario y una contraseña predefinidos**. Solo permite el acceso si las credenciales coinciden con los valores "admin" y "1234".
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtUsuario`
        - **Utilización:** Campo de texto donde el usuario ingresa su **nombre de usuario**.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener la entrada del usuario, la cual se compara con el valor predefinido `"admin"`.
    - **TextBox:**
        - **Name:** `txtContraseña`
        - **PasswordChar:** `*`
        - **Utilización:** Campo de texto donde el usuario ingresa su **contraseña**. La propiedad `PasswordChar` oculta los caracteres que el usuario escribe, reemplazándolos por un asterisco, lo cual es una medida estándar de seguridad.
        - **Propiedades y su uso:** La propiedad `Text` contendrá la contraseña ingresada, que se compara con el valor predefinido `"1234"`.
    - **Button:**
        - **Name:** `btnLogin`
        - **Text:** `"Login"`
        - **Utilización:** Es el botón que el usuario presiona para **intentar iniciar sesión** en el sistema.
        - **Propiedades y su uso:** El evento `Click` (`btnLogin_Click`) es el disparador principal del programa. Al hacer clic, se ejecuta el código que verifica las credenciales. La propiedad `Text` (`"Login"`) proporciona una etiqueta clara para la acción del botón.
- **Explicación:** Al hacer clic en el botón `btnLogin`, se activa el evento `btnLogin_Click`. El código dentro de este evento compara el texto ingresado en `txtUsuario.Text` con la cadena `"admin"` y el texto de `txtContraseña.Text` con la cadena `"1234"`.
    - Si ambas condiciones se cumplen (es decir, el usuario es "admin" y la contraseña es "1234"), se considera que el inicio de sesión es exitoso y se muestra un `MessageBox` con el mensaje **"Login exitoso"**.
    - En caso contrario, si las credenciales no coinciden con los valores predefinidos, se muestra un `MessageBox` con el mensaje **"Usuario o contraseña incorrectos"**.

Programa 281. Sistema médico básico

- **¿En qué consiste el ejercicio?** Este programa implementa un **sistema médico simple que permite registrar pacientes y doctores** en un control `ListBox` para su visualización.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtPaciente`
        - **Utilización:** Se usa para que el usuario ingrese el **nombre del paciente** que desea registrar.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del paciente introducido. Después de agregar el registro, `txtPaciente.Clear()` se usa para limpiar el campo.
    - **TextBox:**
        - **Name:** `txtDoctor`
        - **Utilización:** Se usa para que el usuario ingrese el **nombre del doctor** a registrar.
        - **Propiedades y su uso:** La propiedad `Text` se usa para obtener el nombre del doctor. Después de agregar el registro, `txtDoctor.Clear()` se usa para limpiar el campo.
    - **Button:**
        - **Name:** `btnAgregar`
        - **Text:** `"Agregar Registro"`
        - **Utilización:** Es el botón que el usuario presiona para **añadir el paciente y el doctor** a la lista de registros.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregar_Click`) es el disparador. Al hacer clic, se ejecuta el código que concatena la información y la añade al `ListBox`. La propiedad `Text` (`"Agregar Registro"`) proporciona una etiqueta clara para la acción del botón.
    - **ListBox:**
        - **Name:** `lstRegistros`
        - **Utilización:** Se usa para **mostrar los registros de pacientes y doctores** que se han agregado al sistema.
        - **Propiedades y su uso:** La colección `Items` del `ListBox` es donde se añade la entrada del registro. `lstRegistros.Items.Add()` se utiliza para añadir una nueva entrada de texto, combinando el nombre del paciente y el doctor, por ejemplo, "Paciente: X - Doctor: Y".
- **Explicación:** Cuando el usuario hace clic en el botón `btnAgregar`, se activa el evento `btnAgregar_Click`. Dentro de este evento, el programa toma el texto de `txtPaciente` y `txtDoctor`, los concatena en una cadena de texto descriptiva (como "Paciente: [Nombre Paciente] - Doctor: [Nombre Doctor]"), y luego añade esta cadena como un nuevo elemento a la colección `Items` del `lstRegistros`. Después de añadir el elemento, los campos `txtPaciente` y `txtDoctor` se limpian automáticamente para facilitar la entrada de nuevos registros.

Programa 282. Sistema médico con registro de pacientes

- **¿En qué consiste el ejercicio?** Este programa permite al usuario **registrar nuevos pacientes con su nombre y edad** en una lista.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtNombrePaciente`
        - **Utilización:** Campo de texto donde el usuario introduce el **nombre del paciente**.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del paciente. Después de agregar, `txtNombrePaciente.Clear()` limpia el campo.
    - **TextBox:**
        - **Name:** `txtEdad`
        - **Utilización:** Campo de texto donde el usuario introduce la **edad del paciente**.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener la edad. Después de agregar, `txtEdad.Clear()` limpia el campo.
    - **Button:**
        - **Name:** `btnAgregarPaciente`
        - **Text:** `"Agregar Paciente"`
        - **Utilización:** El botón que el usuario presiona para **añadir la información del paciente** a la lista.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregarPaciente_Click`) es el que contiene la lógica para combinar y añadir los datos a la lista.
    - **ListBox:**
        - **Name:** `lstPacientes`
        - **Utilización:** Muestra los **pacientes registrados**, incluyendo su nombre y edad.
        - **Propiedades y su uso:** `lstPacientes.Items.Add()` se utiliza para añadir una cadena formateada con "Nombre: X - Edad: Y" a la lista.
- **Explicación:** Cuando el usuario hace clic en el botón `btnAgregarPaciente`, se activa el evento `btnAgregarPaciente_Click`. El código toma el texto de `txtNombrePaciente` y `txtEdad`, los concatena en una cadena de texto descriptiva (como "Nombre: [Nombre del Paciente] - Edad: [Edad del Paciente]"), y esta cadena se añade como un nuevo elemento a `lstPacientes.Items`. Finalmente, los campos `txtNombrePaciente` y `txtEdad` se limpian para la siguiente entrada.

Programa 283. Sistema médico con registro de doctores

- **¿En qué consiste el ejercicio?** Este programa permite **registrar nuevos doctores con su nombre y la especialidad** que tienen en una lista.
- **Controles necesarios en el formulario:**

    - **TextBox:**
        - **Name:** `txtNombreDoctor`
        - **Utilización:** Campo de texto donde el usuario introduce el **nombre del doctor**.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del doctor. Después de agregar, `txtNombreDoctor.Clear()` limpia el campo.
    - **TextBox:**
        - **Name:** `txtEspecialidad`
        - **Utilización:** Campo de texto donde el usuario introduce la **especialidad del doctor**.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener la especialidad. Después de agregar, `txtEspecialidad.Clear()` limpia el campo.
    - **Button:**
        - **Name:** `btnAgregarDoctor`
        - **Text:** `"Agregar Doctor"`
        - **Utilización:** El botón que el usuario presiona para **añadir la información del doctor** a la lista.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregarDoctor_Click`) es el que contiene la lógica para combinar y añadir los datos a la lista.
    - **ListBox:**
        - **Name:** `lstDoctores`
        - **Utilización:** Muestra los **doctores registrados**, incluyendo su nombre y la especialidad que imparten.
        - **Propiedades y su uso:** `lstDoctores.Items.Add()` se utiliza para añadir una cadena formateada con "Doctor: X - Especialidad: Y" a la lista.
- **Explicación:** Cuando el usuario hace clic en el botón `btnAgregarDoctor`, se activa el evento `btnAgregarDoctor_Click`. El código toma el texto de `txtNombreDoctor` y `txtEspecialidad`, los combina en una cadena de texto descriptiva (como "Doctor: [Nombre del Doctor] - Especialidad: [Nombre de la Especialidad]"), y esta cadena se añade como un nuevo elemento a `lstDoctores.Items`. Finalmente, los campos `txtNombreDoctor` y `txtEspecialidad` se limpian para la siguiente entrada.

Programa 284. Sistema médico con registro de citas

- **¿En qué consiste el ejercicio?** Este programa permite **registrar citas para pacientes con doctores en una fecha específica** utilizando un control `DateTimePicker`.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtPaciente`
        - **Utilización:** Campo para que el usuario ingrese el **nombre del paciente** para la cita.

        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del paciente. Después de agregar, `txtPaciente.Clear()` limpia el campo.
    - **TextBox:**
        - **Name:** `txtDoctor`
        - **Utilización:** Campo para que el usuario ingrese el **nombre del doctor** para la cita.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del doctor. Después de agregar, `txtDoctor.Clear()` limpia el campo.
    - **DateTimePicker:**
        - **Name:** `dtpFecha`
        - **Utilización:** Permite al usuario **seleccionar la fecha** de la cita.
        - **Propiedades y su uso:** La propiedad `Value` obtiene la fecha y hora seleccionadas. `ToShortDateString()` se utiliza para formatear esta fecha a una cadena corta y legible (ej. "DD/MM/AAAA").
    - **Button:**
        - **Name:** `btnAgregarCita`
        - **Text:** `"Agregar Cita"`
        - **Utilización:** Es el botón que el usuario presiona para **añadir la cita** al sistema.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregarCita_Click`) es el disparador. Al hacer clic, se ejecuta el código que concatena la información y la añade al `ListBox`.
    - **ListBox:**
        - **Name:** `lstCitas`
        - **Utilización:** Se usa para **mostrar las citas registradas**, incluyendo el paciente, el doctor y la fecha.
        - **Propiedades y su uso:** `lstCitas.Items.Add()` se utiliza para añadir una cadena formateada con "Paciente: X - Doctor: Y - Fecha: Z" a la lista.
- **Explicación:** Cuando el usuario hace clic en el botón `btnAgregarCita`, se activa el evento `btnAgregarCita_Click`. El programa toma el texto de `txtPaciente`, `txtDoctor` y la fecha seleccionada en `dtpFecha` (formateada como cadena corta con `dtpFecha.Value.ToShortDateString()`). Toda esta información se combina en una cadena de texto descriptiva (por ejemplo, "Paciente: [Nombre Paciente] - Doctor: [Nombre Doctor] - Fecha: [Fecha de Cita]"). Esta cadena se añade como un nuevo elemento a la colección `Items` del `lstCitas`. Finalmente, los campos `txtPaciente` y `txtDoctor` se limpian automáticamente para facilitar la entrada de nuevas citas.

Programa 285. Sistema médico con historial clínico

- **¿En qué consiste el ejercicio?** Este programa permite **agregar registros al historial clínico de un paciente**, incluyendo un diagnóstico, en una lista.
- **Controles necesarios en el formulario:**

    - **TextBox:**
        - **Name:** `txtPaciente`
        - **Utilización:** Campo para que el usuario ingrese el **nombre del paciente** al que se añadirá el historial.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del paciente. Después de agregar, `txtPaciente.Clear()` limpia el campo.
    - **TextBox:**
        - **Name:** `txtDiagnostico`
        - **Utilización:** Campo para que el usuario ingrese el **diagnóstico** o el registro del historial.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el diagnóstico. Después de agregar, `txtDiagnostico.Clear()` limpia el campo.
    - **Button:**
        - **Name:** `btnAgregarHistorial`
        - **Text:** `"Agregar Historial"`
        - **Utilización:** Es el botón que el usuario presiona para **añadir el registro al historial clínico**.
        - **Propiedades y su uso:** El evento `Click` (`btnAgregarHistorial_Click`) es el disparador. Al hacer clic, se ejecuta el código que concatena la información y la añade al `ListBox`.
    - **ListBox:**
        - **Name:** `lstHistorial`
        - **Utilización:** Se usa para **mostrar los registros del historial clínico**, incluyendo el paciente y su diagnóstico.
        - **Propiedades y su uso:** `lstHistorial.Items.Add()` se utiliza para añadir una cadena formateada con "Paciente: X - Diagnóstico: Y" a la lista.
- **Explicación:** Al hacer clic en el botón `btnAgregarHistorial`, se activa el evento `btnAgregarHistorial_Click`. El programa toma el texto de `txtPaciente` y `txtDiagnostico`, los concatena en una cadena de texto descriptiva (por ejemplo, "Paciente: [Nombre Paciente] - Diagnóstico: [Diagnóstico]"), y luego añade esta cadena como un nuevo elemento a la colección `Items` del `lstHistorial`. Finalmente, los campos `txtPaciente` y `txtDiagnostico` se limpian automáticamente para facilitar la entrada de nuevos registros.

Programa 286. Sistema médico con facturación

- **¿En qué consiste el ejercicio?** Este programa permite **generar una factura para un paciente**, mostrando el monto total, en una lista.
- **Controles necesarios en el formulario:**
    - **TextBox:**
        - **Name:** `txtPaciente`

        - **Utilización:** Campo para que el usuario ingrese el **nombre del paciente** para quien se genera la factura.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del paciente. Después de facturar, `txtPaciente.Clear()` limpia el campo.
    - **TextBox:**
        - **Name:** `txtMonto`
        - **Utilización:** Campo para que el usuario ingrese el **monto total** de la factura.
        - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el monto. Después de facturar, `txtMonto.Clear()` limpia el campo.
    - **Button:**
        - **Name:** `btnFacturar`
        - **Text:** `"Generar Factura"`
        - **Utilización:** Es el botón que el usuario presiona para **generar la factura** y añadirla a la lista.
        - **Propiedades y su uso:** El evento `Click` (`btnFacturar_Click`) es el disparador. Al hacer clic, se ejecuta el código que concatena la información y la añade al `ListBox`.
    - **ListBox:**
        - **Name:** `lstFacturas`
        - **Utilización:** Se usa para **mostrar las facturas generadas**, incluyendo el paciente y el monto.
        - **Propiedades y su uso:** `lstFacturas.Items.Add()` se utiliza para añadir una cadena formateada con "Paciente: X - Monto: $Y" a la lista.
- **Explicación:** Cuando el usuario hace clic en el botón `btnFacturar`, se activa el evento `btnFacturar_Click`. El programa toma el texto de `txtPaciente` y `txtMonto`, los combina en una cadena de texto descriptiva (por ejemplo, "Paciente: [Nombre Paciente] - Monto: $[Monto]"), y luego añade esta cadena como un nuevo elemento a la colección `Items` del `lstFacturas`. Finalmente, los campos `txtPaciente` y `txtMonto` se limpian automáticamente para facilitar la entrada de nuevas facturas.

Programa 287. Sistema médico con reportes

- **¿En qué consiste el ejercicio?** Este programa genera un **reporte simple del historial clínico** al recopilar todos los elementos actualmente listados en un control `ListBox` y mostrarlos en un cuadro de mensaje (`MessageBox`).
- **Controles necesarios en el formulario:**
    - **Button:**
        - **Name:** `btnReporte`
        - **Text:** `"Generar Reporte"`
        - **Utilización:** Es el botón que el usuario presiona para **crear y visualizar el reporte** del historial clínico.

        - **Propiedades y su uso:** El evento `Click` (`btnReporte_Click`) desencadena la recopilación de los datos y la construcción del mensaje del reporte. La propiedad `Text` (`"Generar Reporte"`) etiqueta el botón.
    - **ListBox:**
        - **Name:** `lstHistorial`
        - **Utilización:** Contiene la **lista de registros del historial clínico** que conformarán el reporte.
        - **Propiedades y su uso:** La colección `Items` se itera (`For Each item In lstHistorial.Items`) para obtener cada elemento del historial. `item.ToString()` convierte cada elemento a su representación de cadena para ser incluido en el reporte.
- **Explicación:** Al hacer clic en el botón `btnReporte`, se ejecuta el evento `btnReporte_Click`. El código inicializa una cadena `reporte` con un título "Reporte de Historial Clínico:" y un salto de línea (`Environment.NewLine`). Luego, un bucle `For Each` recorre cada `item` en `lstHistorial.Items`. En cada iteración, el `item` se convierte a cadena y se concatena al `reporte`, seguido de otro salto de línea. Una vez que el bucle ha procesado todos los elementos, el `reporte` completo se muestra al usuario en un `MessageBox`.

Programa 288. Sistema médico con gráficos

- **¿En qué consiste el ejercicio?** Este programa visualiza las citas registradas mediante un **gráfico de columnas** que se muestra en un control `Chart`.
- **Controles necesarios en el formulario:**
    - **Chart:**
        - **Name:** `chartCitas`
        - **ChartType:** `Column` (configurado en tiempo de ejecución)
        - **Utilización:** Control gráfico que **muestra visualmente los datos** de las citas.
        - **Propiedades y su uso:** La propiedad `Series` es una colección donde se añaden los conjuntos de datos. `chartCitas.Series.Clear()` limpia series anteriores. `ChartType` de una serie se establece a `DataVisualization.Charting.SeriesChartType.Column` para que se muestre como un gráfico de columnas. `Points.AddXY(item.ToString(), 1)` añade puntos de datos, usando la descripción de la cita como etiqueta X y un valor fijo de 1 para representar su existencia.
    - **Button:**
        - **Name:** `btnGraficar`
        - **Text:** `"Mostrar Gráfico"`
        - **Utilización:** Botón para **generar y actualizar el gráfico** de citas.
        - **Propiedades y su uso:** El evento `Click` (`btnGraficar_Click`) es el que contiene la lógica para construir y mostrar el gráfico.
    - **ListBox:**

        - **Name:** `lstCitas`
        - **Utilización:** Contiene la **lista de citas** que se utilizarán como datos para el gráfico.
        - **Propiedades y su uso:** La colección `Items` de `lstCitas` se itera (`For Each item In lstCitas.Items`) para obtener cada elemento de cita y añadirlo al gráfico.
- **Explicación:** Al hacer clic en el botón `btnGraficar`, se activa el evento `btnGraficar_Click`. El código comienza por **limpiar cualquier serie de datos existente** en `chartCitas` con `chartCitas.Series.Clear()`. Luego, se crea una nueva instancia de `Series` y se nombra "Citas". Se establece el `ChartType` de esta serie como `Column`. A continuación, un bucle `For Each` recorre cada `item` en la colección `lstCitas.Items`. Por cada `item`, se añade un punto al gráfico utilizando `serie.Points.AddXY(item.ToString(), 1)`. Esto significa que cada cita del `ListBox` aparecerá como una columna en el gráfico, con la descripción de la cita como etiqueta y una altura de 1 (simbolizando su existencia). Finalmente, esta `serie` se añade a la colección `chartCitas.Series`, lo que hace que el gráfico se dibuje y se muestre en el formulario. **Requisito adicional:** Para utilizar el control `Chart`, es necesario que la referencia `System.Windows.Forms.DataVisualization` esté añadida al proyecto.

Programa 289. Sistema médico con exportación Excel

- **¿En qué consiste el ejercicio?** Este programa es un **ejemplo conceptual que simula la exportación de datos médicos a una hoja de cálculo de Microsoft Excel**. No contiene una implementación real de la exportación, sino que indica lo que se necesitaría.
- **Controles necesarios en el formulario:**
    - **Button:**
        - **Name:** `btnExportar`
        - **Text:** `"Exportar a Excel"`
        - **Utilización:** Botón que el usuario presiona para **iniciar el proceso conceptual de exportación**.
        - **Propiedades y su uso:** El evento `Click` (`btnExportar_Click`) es el único controlador de eventos en este programa, y simplemente muestra un mensaje. La propiedad `Text` (`"Exportar a Excel"`) etiqueta el botón.
- **Explicación:** Cuando el usuario hace clic en el botón `btnExportar`, se ejecuta el evento `btnExportar_Click`. El código dentro de este evento es muy simple y consiste en una única línea que muestra un `MessageBox` con el mensaje: **"Datos exportados a Excel (requiere referencia a Microsoft Excel)"**. La fuente aclara que, para una implementación real de esta funcionalidad, se necesitaría agregar una referencia a la librería de objetos de Microsoft Excel en el proyecto. Este programa se limita a ilustrar el concepto sin el código complejo de interacción con Excel.

Programa 290. Sistema médico con importación Excel

- **¿En qué consiste el ejercicio?** Este programa es un **ejemplo conceptual que simula la importación de datos médicos desde una hoja de cálculo de Microsoft Excel**. Al igual que el de exportación, se centra en la idea más que en la implementación completa.
- **Controles necesarios en el formulario:**
    - **Button:**
        - **Name:** `btnImportar`
        - **Text:** `"Importar desde Excel"`
        - **Utilización:** Botón que el usuario presiona para **iniciar el proceso conceptual de importación**.
        - **Propiedades y su uso:** El evento `Click` (`btnImportar_Click`) es el único controlador de eventos y solo muestra un mensaje informativo. La propiedad `Text` (`"Importar desde Excel"`) etiqueta el botón.
- **Explicación:** Al hacer clic en el botón `btnImportar`, se activa el evento `btnImportar_Click`. El código consiste en una línea que muestra un `MessageBox` con el mensaje: "**Datos importados desde Excel (requiere referencia a Microsoft Excel)**". Similar al programa de exportación, la fuente indica que para una implementación funcional, se requeriría agregar una referencia a la librería de objetos de Microsoft Excel en el proyecto. El programa, por lo tanto, es un marcador de posición para una funcionalidad más compleja.

Programa 291. Sistema médico con múltiples usuarios

- **¿En qué consiste el ejercicio?** Este programa es un ejemplo **conceptual que simula el inicio de sesión para diferentes tipos de usuarios** (Admin, Doctor, Recepción) dentro de un sistema médico, donde el tipo de usuario se selecciona a través de un control `ComboBox`. No incluye una autenticación real de credenciales.
- **Controles necesarios en el formulario:**
    - **ComboBox:**
        - **Name:** `cmbUsuarios`
        - **Items:** `"Admin"`, `"Doctor"`, `"Recepción"`
        - **Utilización:** Permite al usuario **seleccionar su rol o tipo de usuario** antes de "acceder" al sistema.
        - **Propiedades y su uso:** La propiedad `SelectedItem` del `ComboBox` se utiliza para obtener la cadena de texto que representa el tipo de usuario elegido por el usuario (`"Admin"`, `"Doctor"` o `"Recepción"`). Esta selección se usa para el mensaje conceptual de inicio de sesión.
    - **Button:**
        - **Name:** `btnAcceder`
        - **Text:** `"Acceder"`
        - **Utilización:** Es el botón que el usuario presiona para **simular el acceso al sistema** con el rol seleccionado.

        - **Propiedades y su uso:** El evento `Click` (`btnAcceder_Click`) es el disparador. Al hacer clic, se ejecuta el código que muestra un `MessageBox` indicando el tipo de usuario que ha "iniciado sesión". La propiedad `Text` (`"Acceder"`) proporciona una etiqueta clara para la acción.
- **Explicación:** Cuando el usuario hace clic en el botón `btnAcceder`, se activa el evento `btnAcceder_Click`. El código dentro de este evento recupera el tipo de usuario seleccionado del `cmbUsuarios.SelectedItem`. Luego, muestra un **`MessageBox` que indica conceptualmente que el "Usuario: [Tipo de Usuario] ha iniciado sesión"**. Este programa es una demostración conceptual y no realiza ninguna validación de credenciales real ni autenticación contra un sistema de usuarios.

Programa 292. Sistema médico con permisos

- **¿En qué consiste el ejercicio?** Este programa demuestra cómo la acción de **"eliminar registro" puede estar restringida a ciertos roles de usuario**, como "Admin", basándose en la selección de un control `ComboBox`. Este es un ejemplo de control de acceso básico basado en roles.
- **Controles necesarios en el formulario:**
    - **ComboBox:**
        - **Name:** `cmbRoles`
        - **Items:** `"Admin"`, `"Doctor"`
        - **Utilización:** Permite al usuario **seleccionar el rol o nivel de permiso** con el que desea realizar la acción de eliminación.
        - **Propiedades y su uso:** La propiedad `SelectedItem` se utiliza para obtener la cadena que representa el rol elegido por el usuario (`"Admin"` o `"Doctor"`). Este valor es crucial para la lógica de verificación de permisos.
    - **Button:**
        - **Name:** `btnEliminarRegistro`
        - **Text:** `"Eliminar Registro"`
        - **Utilización:** El botón que el usuario presiona para **intentar ejecutar la acción de eliminar un registro**.
        - **Propiedades y su uso:** El evento `Click` (`btnEliminarRegistro_Click`) es donde se ejecuta la lógica de verificación de permisos. La propiedad `Text` (`"Eliminar Registro"`) proporciona una etiqueta clara para la acción.
- **Explicación:** Cuando el usuario hace clic en el botón `btnEliminarRegistro`, se activa el evento `btnEliminarRegistro_Click`. El código dentro de este evento **verifica el rol seleccionado en el `cmbRoles.SelectedItem`**.
    - Si el rol seleccionado es **"Admin"**, se muestra un `MessageBox` con el mensaje **"Registro eliminado"**, simulando que la acción se ha realizado con éxito debido a los permisos adecuados.

- Si el rol seleccionado **no es "Admin"** (por ejemplo, si se selecciona "Doctor"), se muestra un `MessageBox` indicando **"No tiene permisos para eliminar"**, denegando la acción.

Programa 293. Sistema médico con auditoría

- **¿En qué consiste el ejercicio?** Este programa implementa un mecanismo de auditoría al **registrar cada adición de un paciente en un control `ListBox` específico**, incluyendo la fecha y hora exactas del evento, así como el nombre del paciente agregado.
- **Controles necesarios en el formulario:**
  - **TextBox:**
    - **Name:** `txtPaciente`
    - **Utilización:** Se usa para que el usuario ingrese el **nombre del paciente** que desea registrar en el sistema médico.
    - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener el nombre del paciente introducido por el usuario. Después de registrar la acción, `txtPaciente.Clear()` se usa para limpiar el campo, preparándolo para la siguiente entrada.
  - **Button:**
    - **Name:** `btnAgregarPaciente`
    - **Text:** `"Agregar Paciente"`
    - **Utilización:** Es el botón que el usuario presiona para **añadir un nuevo paciente** y, consecuentemente, generar una entrada en el registro de auditoría.
    - **Propiedades y su uso:** El evento `Click` (`btnAgregarPaciente_Click`) es el disparador principal del programa. Al hacer clic, se ejecuta el código que registra la información de auditoría. La propiedad `Text` (`"Agregar Paciente"`) proporciona una etiqueta clara para la acción del botón.
  - **ListBox:**
    - **Name:** `lstAuditoria`
    - **Utilización:** Se usa para **mostrar los registros de auditoría**, listando cada vez que un paciente es agregado.
    - **Propiedades y su uso:** La colección `Items` de este `ListBox` es fundamental. `lstAuditoria.Items.Add()` se utiliza para añadir una nueva entrada de texto formateada con la fecha, hora y el nombre del paciente que fue agregado.
- **Explicación:** Cuando el usuario hace clic en el botón `btnAgregarPaciente`, se activa el evento `btnAgregarPaciente_Click`. El código dentro de este evento construye una cadena de auditoría que incluye la **fecha y hora actuales** (obtenida mediante `DateTime.Now.ToString()`) y una descripción del evento: " - Paciente agregado: [nombre del paciente]". Esta cadena se añade como un nuevo elemento a la colección `Items` del `lstAuditoria`, haciendo visible el registro. Finalmente, el campo `txtPaciente` se limpia automáticamente para la siguiente entrada.

Programa 294. Sistema médico con conexión remota

- **¿En qué consiste el ejercicio?** Este programa es un **ejemplo conceptual que simula la conexión a un servidor remoto** para la gestión del sistema médico, sin implementar la complejidad real de una conexión de red.
- **Controles necesarios en el formulario:**
    - **Button:**
        - **Name:** `btnConectar`
        - **Text:** `"Conectar a Servidor"`
        - **Utilización:** Es el botón que el usuario presiona para **simular el intento de conexión** a un servidor remoto.
        - **Propiedades y su uso:** El evento `Click` (`btnConectar_Click`) es el único controlador de eventos en este programa. Al ser clicado, simplemente muestra un mensaje informativo. La propiedad `Text` (`"Conectar a Servidor"`) etiqueta el botón con su acción.
- **Explicación:** Al hacer clic en el botón `btnConectar`, se ejecuta el evento `btnConectar_Click`. El código dentro de este evento es muy simple y consiste en una única línea que muestra un **`MessageBox` con el mensaje: "Conexión remota establecida (simulada)"**. La explicación del programa aclara que este es meramente un concepto y que una implementación real de una conexión remota en un sistema médico requeriría un código de red mucho más complejo y una infraestructura de servidor, que no están incluidos en este ejemplo.

Programa 295. Sistema médico con backup automático

- **¿En qué consiste el ejercicio?** Este programa es un **ejemplo conceptual que simula la realización de un backup automático** de los datos del sistema médico. No realiza un backup real, sino que conceptualiza la idea.
- **Controles necesarios en el formulario:**
    - **Button:**
        - **Name:** `btnBackup`
        - **Text:** `"Realizar Backup"`
        - **Utilización:** Es el botón que el usuario presiona para **simular la ejecución de un backup** de los datos médicos.
        - **Propiedades y su uso:** El evento `Click` (`btnBackup_Click`) es el único controlador de eventos en este programa. Al ser clicado, simplemente muestra un mensaje. La propiedad `Text` (`"Realizar Backup"`) etiqueta el botón con su acción.
- **Explicación:** Al hacer clic en el botón `btnBackup`, se ejecuta el evento `btnBackup_Click`. El código dentro de este evento es una única línea que muestra un **`MessageBox` con el mensaje: "Backup automático de datos médicos realizado**

correctamente". Este programa conceptualiza la idea de un backup automático para un sistema médico, pero la implementación real implicaría operaciones más complejas como copiar archivos de bases de datos, exportar datos a formatos específicos o interactuar con herramientas de backup, las cuales no están presentes en este ejemplo.

Programa 296. Sistema médico con restauración de datos

- **¿En qué consiste el ejercicio?** Este programa es un **ejemplo conceptual que simula la restauración de datos** del sistema médico a partir de un backup previamente realizado. No implementa una restauración real, sino que ilustra la idea.
- **Controles necesarios en el formulario:**
    - **Button:**
        - **Name:** `btnRestaurar`
        - **Text:** `"Restaurar Backup"`
        - **Utilización:** Es el botón que el usuario presiona para **simular la restauración de los datos** médicos.
        - **Propiedades y su uso:** El evento `Click` (`btnRestaurar_Click`) es el único controlador de eventos en este programa. Al ser clicado, simplemente muestra un mensaje. La propiedad `Text` (`"Restaurar Backup"`) etiqueta el botón con su acción.
- **Explicación:** Cuando el usuario hace clic en el botón `btnRestaurar`, se ejecuta el evento `btnRestaurar_Click`. El código dentro de este evento es una única línea que muestra un **`MessageBox` con el mensaje: "Datos médicos restaurados desde backup"**. Este programa ilustra conceptualmente la acción de restaurar datos, pero una implementación real requeriría operaciones más complejas como reemplazar archivos de bases de datos, importar datos de un archivo de backup anterior o interactuar con un sistema de gestión de backups.

Programa 297. Sistema médico con notificaciones

- **¿En qué consiste el ejercicio?** Este programa utiliza un control `NotifyIcon` para **mostrar una notificación emergente en la bandeja del sistema (system tray)**, alertando al usuario sobre un "Nuevo evento registrado" en el sistema médico.
- **Controles necesarios en el formulario:**
    - **NotifyIcon:**
        - **Name:** `NotifyIcon1`
        - **Icon:** `icono.ico` (se requiere un archivo de ícono para que el icono de la aplicación sea visible en la bandeja del sistema)
        - **Utilización:** Este control permite a la aplicación mostrar un icono en la bandeja del sistema y **enviar notificaciones tipo "globo"** al usuario.
        - **Propiedades y su uso:** La propiedad `Visible` se establece en `True` (`NotifyIcon1.Visible = True`) para que el icono aparezca en la bandeja del sistema. El método `ShowBalloonTip()` es el principal para

mostrar la notificación, con parámetros que controlan la duración (3000 ms = 3 segundos), el título de la notificación ("Sistema Médico"), el mensaje ("Nuevo evento registrado") y el tipo de icono que acompaña la notificación (`ToolTipIcon.Info` para indicar información).

- **Button:**
  - **Name:** `btnNotificar`
  - **Text:** `"Notificar Evento"`
  - **Utilización:** Es el botón que el usuario presiona para **simular y activar la notificación** de un nuevo evento.
  - **Propiedades y su uso:** El evento `Click` (`btnNotificar_Click`) es el disparador que ejecuta la lógica para mostrar la notificación. La propiedad `Text` (`"Notificar Evento"`) etiqueta el botón con su acción.

- **Explicación:** Al hacer clic en el botón `btnNotificar`, se activa el evento `btnNotificar_Click`. Dentro de este evento, la propiedad `NotifyIcon1.Visible` se establece en `True` para asegurar que el icono de la aplicación aparezca en la bandeja del sistema. Luego, el método `NotifyIcon1.ShowBalloonTip()` se invoca para mostrar la notificación emergente. Esta notificación tiene un título de **"Sistema Médico"**, un mensaje de **"Nuevo evento registrado"**, una duración de 3 segundos y utiliza un icono de información (`ToolTipIcon.Info`).

Programa 298. Sistema médico con recordatorios de citas

- **¿En qué consiste el ejercicio?** Este programa permite al usuario **establecer un recordatorio para una cita**, mostrando un `MessageBox` con la fecha programada seleccionada.

- **Controles necesarios en el formulario:**

  - **DateTimePicker:**
    - **Name:** `dtpCita`
    - **Utilización:** Permite al usuario **seleccionar una fecha** para la cita que desea recordar.
    - **Propiedades y su uso:** La propiedad `Value` de `dtpCita` obtiene la fecha y hora seleccionadas. `ToShortDateString()` se utiliza para formatear esta fecha a una cadena corta y legible (ej. "DD/MM/AAAA") para su visualización en el mensaje.
  - **Button:**
    - **Name:** `btnRecordatorio`
    - **Text:** `"Recordar Cita"`
    - **Utilización:** Es el botón que el usuario presiona para **activar la visualización del recordatorio** de la cita.
    - **Propiedades y su uso:** El evento `Click` (`btnRecordatorio_Click`) es el disparador. Al hacer clic, se ejecuta el código que construye y muestra el mensaje de recordatorio. La propiedad `Text` (`"Recordar Cita"`) etiqueta el botón.

- **Explicación:** Cuando el usuario hace clic en el botón `btnRecordatorio`, se activa el evento `btnRecordatorio_Click`. El código dentro de este evento recupera la fecha seleccionada en `dtpCita.Value`, la formatea a una cadena corta (`ToShortDateString()`), y luego muestra un **`MessageBox` con el mensaje: "Recordatorio: Cita programada para el [fecha seleccionada]"**.

Programa 299. Sistema médico con estadísticas

- **¿En qué consiste el ejercicio?** Este programa proporciona **estadísticas básicas del sistema médico** al mostrar el número total de pacientes actualmente registrados en un `ListBox` mediante un cuadro de mensaje (`MessageBox`).
- **Controles necesarios en el formulario:**
    - **ListBox:**
        - **Name:** `lstPacientes`
        - **Utilización:** Contiene la **lista de pacientes** del sistema médico. Es la fuente de datos para la estadística.
        - **Propiedades y su uso:** La propiedad `Items.Count` de `lstPacientes` es crucial, ya que devuelve el número de elementos que hay en la lista, lo que se interpreta como el total de pacientes registrados.
    - **Button:**
        - **Name:** `btnEstadisticas`
        - **Text:** `"Mostrar Estadísticas"`
        - **Utilización:** Es el botón que el usuario presiona para **obtener y visualizar las estadísticas** del sistema médico.
        - **Propiedades y su uso:** El evento `Click` (`btnEstadisticas_Click`) es el disparador. Al hacer clic, se ejecuta el código que calcula y muestra el total de pacientes. La propiedad `Text` (`"Mostrar Estadísticas"`) etiqueta el botón.
- **Explicación:** Cuando el usuario hace clic en el botón `btnEstadisticas`, se activa el evento `btnEstadisticas_Click`. Dentro de este evento, el programa obtiene el número de elementos presentes en `lstPacientes` utilizando `lstPacientes.Items.Count` y lo almacena en la variable `totalPacientes`. Finalmente, un **`MessageBox` se encarga de mostrar al usuario esta estadística, indicando "Total de pacientes registrados: [número]"**.

Programa 300. Sistema médico con login de usuarios

- **¿En qué consiste el ejercicio?** Este programa implementa un **sistema de login básico para el sistema médico con un nombre de usuario y una contraseña predefinidos**. Solo permite el acceso si las credenciales coinciden con los valores "admin" y "1234".
- **Controles necesarios en el formulario:**

- **TextBox:**
  - **Name:** `txtUsuario`
  - **Utilización:** Campo de texto donde el usuario ingresa su **nombre de usuario**.
  - **Propiedades y su uso:** La propiedad `Text` se utiliza para obtener la entrada del usuario, la cual se compara con el valor predefinido `"admin"`.
- **TextBox:**
  - **Name:** `txtContraseña`
  - **PasswordChar:** `*`
  - **Utilización:** Campo de texto donde el usuario ingresa su **contraseña**. La propiedad `PasswordChar` oculta los caracteres que el usuario escribe, reemplazándolos por un asterisco, lo cual es una medida estándar de seguridad.
  - **Propiedades y su uso:** La propiedad `Text` contendrá la contraseña ingresada, que se compara con el valor predefinido `"1234"`.
- **Button:**
  - **Name:** `btnLogin`
  - **Text:** `"Login"`
  - **Utilización:** Es el botón que el usuario presiona para **intentar iniciar sesión** en el sistema.
  - **Propiedades y su uso:** El evento `Click` (`btnLogin_Click`) es el disparador principal del programa. Al hacer clic, se ejecuta el código que verifica las credenciales. La propiedad `Text` (`"Login"`) proporciona una etiqueta clara para la acción del botón.

- **Explicación:** Al hacer clic en el botón `btnLogin`, se activa el evento `btnLogin_Click`. El código dentro de este evento compara el texto ingresado en `txtUsuario.Text` con la cadena `"admin"` y el texto de `txtContraseña.Text` con la cadena `"1234"`.
  - Si ambas condiciones se cumplen (es decir, el usuario es "admin" y la contraseña es "1234"), se considera que el inicio de sesión es exitoso y se muestra un **`MessageBox` con el mensaje "Login exitoso"**.
  - En caso contrario, si las credenciales no coinciden con los valores predefinidos, se muestra un **`MessageBox` con el mensaje "Usuario o contraseña incorrectos"**.

Ejercicios

Ejercicios de VB Net Form

Programa 1. Formulario con un botón que muestra un mensaje

Controles:

- Button → Name: `btnMensaje`, Text: "Mostrar Mensaje"

Código:

```
Public Class Form1
    Private Sub btnMensaje_Click(sender As Object, e As EventArgs) Handles
btnMensaje.Click
        MessageBox.Show("Hola, este es un mensaje")
    End Sub
End Class
```

Programa 2. Formulario con una etiqueta y un botón que cambia el texto

Controles:

- Label → Name: `lblTexto`, Text: "Texto inicial"
- Button → Name: `btnCambiar`, Text: "Cambiar Texto"

Código:

```
Public Class Form1
    Private Sub btnCambiar_Click(sender As Object, e As EventArgs) Handles
btnCambiar.Click
        lblTexto.Text = "Texto cambiado"
    End Sub
End Class
```

Programa 3. Formulario con un cuadro de texto y un botón para mostrar su contenido

Controles:

- TextBox → Name: `txtEntrada`
- Button → Name: `btnMostrar`, Text: "Mostrar"

Código:

```
Public Class Form1
    Private Sub btnMostrar_Click(sender As Object, e As EventArgs) Handles
btnMostrar.Click
        MessageBox.Show("Contenido: " & txtEntrada.Text)
    End Sub
End Class
```

Programa 4. Formulario con dos botones que cambian el color de fondo
Controles:

- Button → Name: `btnRojo`, Text: "Fondo Rojo"
- Button → Name: `btnAzul`, Text: "Fondo Azul"

Código:

```
Public Class Form1
    Private Sub btnRojo_Click(sender As Object, e As EventArgs) Handles
btnRojo.Click
        Me.BackColor = Color.Red
    End Sub

    Private Sub btnAzul_Click(sender As Object, e As EventArgs) Handles
btnAzul.Click
        Me.BackColor = Color.Blue
    End Sub
End Class
```

Programa 5. Formulario con un cuadro de texto que convierte a mayúsculas
Controles:

- TextBox → Name: `txtEntrada`
- Button → Name: `btnMayusculas`, Text: "Convertir"

Código:

```
Public Class Form1
    Private Sub btnMayusculas_Click(sender As Object, e As EventArgs) Handles
btnMayusculas.Click
        txtEntrada.Text = txtEntrada.Text.ToUpper()
    End Sub
End Class
```

Programa 6. Formulario con dos cuadros de texto para sumar números
Controles:

- TextBox → Name: `txtNum1`
- TextBox → Name: `txtNum2`
- Button → Name: `btnSumar`, Text: "Sumar"
- Label → Name: `lblResultado`, Text: "Resultado:"

Código:

```
Public Class Form1
    Private Sub btnSumar_Click(sender As Object, e As EventArgs) Handles
btnSumar.Click
        Dim a As Double = Val(txtNum1.Text)
        Dim b As Double = Val(txtNum2.Text)
        lblResultado.Text = "Resultado: " & (a + b).ToString()
    End Sub
End Class
```

Programa 7. Formulario con un botón para limpiar cuadros de texto
Controles:

- TextBox → Name: `txtUno`
- TextBox → Name: `txtDos`
- Button → Name: `btnLimpiar`, Text: "Limpiar"

Código:

```
Public Class Form1
    Private Sub btnLimpiar_Click(sender As Object, e As EventArgs) Handles
btnLimpiar.Click
        txtUno.Clear()
        txtDos.Clear()
    End Sub
End Class
```

Programa 8. Formulario con un botón que cierra la aplicación
Controles:

- Button → Name: `btnSalir`, Text: "Salir"

Código:

```
Public Class Form1
    Private Sub btnSalir_Click(sender As Object, e As EventArgs) Handles
btnSalir.Click
        Application.Exit()
    End Sub
End Class
```

Programa 9. Formulario con una etiqueta que muestra la hora actual
Controles:

- Label → Name: `lblHora`, Text: "Hora:"
- Timer → Name: `Timer1`, Enabled: True, Interval: 1000

Código:

```
Public Class Form1
    Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
        lblHora.Text = "Hora: " & DateTime.Now.ToLongTimeString()
    End Sub
End Class
```

Programa 10. Formulario con un cuadro de texto que cuenta caracteres
Controles:

- TextBox → Name: `txtEntrada`, Multiline: True
- Label → Name: `lblContar`, Text: "Caracteres: 0"

Código:

```
Public Class Form1
    Private Sub txtEntrada_TextChanged(sender As Object, e As EventArgs) Handles
txtEntrada.TextChanged
        lblContar.Text = "Caracteres: " & txtEntrada.Text.Length.ToString()
    End Sub
End Class
```

Programa 11. Formulario con dos botones para mostrar imágenes diferentes
Controles:

- PictureBox → Name: `picImagen`, SizeMode: StretchImage, Size: 200x200
- Button → Name: `btnImagen1`, Text: "Imagen 1"
- Button → Name: `btnImagen2`, Text: "Imagen 2"

Código:

```
Public Class Form1

    Private Sub btnImagen1_Click(sender As Object, e As EventArgs) Handles
btnImagen1.Click

        picImagen.Image = Image.FromFile("imagen1.jpg")

    End Sub


    Private Sub btnImagen2_Click(sender As Object, e As EventArgs) Handles
btnImagen2.Click

        picImagen.Image = Image.FromFile("imagen2.jpg")

    End Sub

End Class
```

Programa 12. Formulario con un botón que cambia el tamaño de la ventana
Controles:

- Button → Name: `btnCambiarTamaño`, Text: "Cambiar Tamaño"

Código:

```
Public Class Form1

    Private Sub btnCambiarTamaño_Click(sender As Object, e As EventArgs) Handles
btnCambiarTamaño.Click

        Me.Size = New Size(800, 600)

    End Sub

End Class
```

Programa 13. Formulario con un cuadro de texto que solo permite números
Controles:

- TextBox → Name: `txtNumeros`

Código:

```
Public Class Form1

    Private Sub txtNumeros_KeyPress(sender As Object, e As KeyPressEventArgs)
Handles txtNumeros.KeyPress

        If Not Char.IsControl(e.KeyChar) AndAlso Not Char.IsDigit(e.KeyChar)
Then

            e.Handled = True

        End If

    End Sub

End Class
```

Programa 14. Formulario con tres botones que cambian el color de texto
Controles:

- Label → Name: `lblTexto`, Text: "Texto de ejemplo"
- Button → Name: `btnRojo`, Text: "Rojo"
- Button → Name: `btnVerde`, Text: "Verde"
- Button → Name: `btnAzul`, Text: "Azul"

Código:

```
Public Class Form1

    Private Sub btnRojo_Click(sender As Object, e As EventArgs) Handles
btnRojo.Click

        lblTexto.ForeColor = Color.Red

    End Sub


    Private Sub btnVerde_Click(sender As Object, e As EventArgs) Handles
btnVerde.Click

        lblTexto.ForeColor = Color.Green

    End Sub


    Private Sub btnAzul_Click(sender As Object, e As EventArgs) Handles
btnAzul.Click

        lblTexto.ForeColor = Color.Blue

    End Sub

End Class
```

Programa 15. Formulario con un cuadro de texto y un botón para invertir texto
Controles:

- TextBox → Name: `txtEntrada`
- Button → Name: `btnInvertir`, Text: "Invertir"

Código:

```
Public Class Form1

    Private Sub btnInvertir_Click(sender As Object, e As EventArgs) Handles
btnInvertir.Click

        Dim texto As String = txtEntrada.Text

        Dim invertido As String = StrReverse(texto)

        txtEntrada.Text = invertido

    End Sub

End Class
```

Programa 16. Formulario con un contador de clics
Controles:

- Label → Name: `lblContador`, Text: "Clics: 0"
- Button → Name: `btnContar`, Text: "Contar Clic"

Código:

```
Public Class Form1

    Dim contador As Integer = 0


    Private Sub btnContar_Click(sender As Object, e As EventArgs) Handles
btnContar.Click

        contador += 1

        lblContador.Text = "Clics: " & contador.ToString()

    End Sub

End Class
```

Programa 17. Formulario con dos botones para mostrar mensajes distintos
Controles:

- Button → Name: `btnMensaje1`, Text: "Mensaje 1"
- Button → Name: `btnMensaje2`, Text: "Mensaje 2"

Código:

```
Public Class Form1

    Private Sub btnMensaje1_Click(sender As Object, e As EventArgs) Handles
btnMensaje1.Click

        MessageBox.Show("Este es el primer mensaje")

    End Sub


    Private Sub btnMensaje2_Click(sender As Object, e As EventArgs) Handles
btnMensaje2.Click

        MessageBox.Show("Este es el segundo mensaje")

    End Sub

End Class
```

Programa 18. Formulario con un CheckBox que habilita o deshabilita un botón
Controles:

- CheckBox → Name: `chkHabilitar`, Text: "Habilitar botón"
- Button → Name: `btnAccion`, Text: "Acción", Enabled: False

Código:

```
Public Class Form1

    Private Sub chkHabilitar_CheckedChanged(sender As Object, e As EventArgs)
Handles chkHabilitar.CheckedChanged

        btnAccion.Enabled = chkHabilitar.Checked

    End Sub

End Class
```

Programa 19. Formulario con un TextBox multilinea y un botón para contar líneas
Controles:

- TextBox → Name: `txtMultilinea`, Multiline: True, Size: 200x100
- Button → Name: `btnContar`, Text: "Contar Líneas"
- Label → Name: `lblLineas`, Text: "Líneas: 0"

Código:

```
Public Class Form1

    Private Sub btnContar_Click(sender As Object, e As EventArgs) Handles
btnContar.Click

        Dim lineas As Integer = txtMultilinea.Lines.Length

        lblLineas.Text = "Líneas: " & lineas.ToString()

    End Sub

End Class
```

Programa 20. Formulario con una etiqueta que cambia de color al pasar el mouse
Controles:

- Label → Name: `lblColor`, Text: "Pasa el mouse por aquí"

Código:

```
Public Class Form1

    Private Sub lblColor_MouseEnter(sender As Object, e As EventArgs) Handles
lblColor.MouseEnter

        lblColor.ForeColor = Color.Red

    End Sub


    Private Sub lblColor_MouseLeave(sender As Object, e As EventArgs) Handles
lblColor.MouseLeave

        lblColor.ForeColor = Color.Black

    End Sub

End Class
```

Programa 21. Formulario con un botón que cambia el ícono del formulario
Controles:

- Button → Name: `btnIcono`, Text: "Cambiar Ícono"

Código:

```
Public Class Form1

    Private Sub btnIcono_Click(sender As Object, e As EventArgs) Handles
btnIcono.Click

        Me.Icon = New Icon("icono.ico")

    End Sub

End Class
```

Programa 22. Formulario con un TrackBar que controla el tamaño del texto
Controles:

- Label → Name: `lblTexto`, Text: "Texto de ejemplo"
- TrackBar → Name: `trkTamaño`, Minimum: 8, Maximum: 40, Value: 12

Código:

```
Public Class Form1

    Private Sub trkTamaño_Scroll(sender As Object, e As EventArgs) Handles
trkTamaño.Scroll

        lblTexto.Font = New Font(lblTexto.Font.FontFamily, trkTamaño.Value)

    End Sub

End Class
```

Programa 23. Formulario con un cuadro de texto que valida correo electrónico
Controles:

- TextBox → Name: `txtCorreo`
- Button → Name: `btnValidar`, Text: "Validar"
- Label → Name: `lblResultado`, Text: ""

Código:

```
Imports System.Text.RegularExpressions


Public Class Form1

    Private Sub btnValidar_Click(sender As Object, e As EventArgs) Handles
btnValidar.Click

        Dim patron As String = "^[^@\s]+@[^@\s]+\.[^@\s]+$"

        If Regex.IsMatch(txtCorreo.Text, patron) Then

            lblResultado.Text = "Correo válido"

        Else

            lblResultado.Text = "Correo inválido"

        End If

    End Sub

End Class
```

Programa 24. Formulario con un botón que rota imágenes
Controles:

- PictureBox → Name: `picImagen`, SizeMode: StretchImage, Size: 200x200
- Button → Name: `btnRotar`, Text: "Rotar"

Código:

```
Public Class Form1
    Dim imagen As Image = Image.FromFile("imagen.jpg")

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        picImagen.Image = imagen
    End Sub

    Private Sub btnRotar_Click(sender As Object, e As EventArgs) Handles
btnRotar.Click
        imagen.RotateFlip(RotateFlipType.Rotate90FlipNone)
        picImagen.Image = imagen
    End Sub
End Class
```

Programa 25. Formulario con una lista de opciones en un ComboBox
Controles:

- ComboBox → Name: `cmbOpciones`, DropDownStyle: DropDownList

Código:

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        cmbOpciones.Items.Add("Opción 1")
        cmbOpciones.Items.Add("Opción 2")
        cmbOpciones.Items.Add("Opción 3")
    End Sub
End Class
```

Programa 26. Formulario con un ListBox que agrega elementos desde un TextBox
Controles:

- TextBox → Name: `txtElemento`
- Button → Name: `btnAgregar`, Text: "Agregar"
- ListBox → Name: `lstElementos`

Código:

```
Public Class Form1

    Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click

        If txtElemento.Text <> "" Then

            lstElementos.Items.Add(txtElemento.Text)

            txtElemento.Clear()

        End If

    End Sub

End Class
```

Programa 27. Formulario con un botón que elimina el elemento seleccionado del ListBox
Controles:

- ListBox → Name: `lstElementos`
- Button → Name: `btnEliminar`, Text: "Eliminar Seleccionado"

Código:

```
Public Class Form1

    Private Sub btnEliminar_Click(sender As Object, e As EventArgs) Handles
btnEliminar.Click

        If lstElementos.SelectedIndex >= 0 Then

            lstElementos.Items.RemoveAt(lstElementos.SelectedIndex)

        End If

    End Sub

End Class
```

Programa 28. Formulario con un cuadro de texto que convierte a número romano
Controles:

- TextBox → Name: `txtNumero`
- Button → Name: `btnConvertir`, Text: "Convertir"
- Label → Name: `lblRomano`, Text: ""

Código:

```
Public Class Form1

    Private Function ConvertirRomano(num As Integer) As String

        Dim valores() As Integer = {1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1}

        Dim simbolos() As String = {"M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I"}

        Dim resultado As String = ""

        For i As Integer = 0 To valores.Length - 1

            While num >= valores(i)

                num -= valores(i)

                resultado &= simbolos(i)

            End While

        Next

        Return resultado

    End Function


    Private Sub btnConvertir_Click(sender As Object, e As EventArgs) Handles btnConvertir.Click

        Dim numero As Integer

        If Integer.TryParse(txtNumero.Text, numero) Then

            lblRomano.Text = ConvertirRomano(numero)

        Else

            lblRomano.Text = "Número inválido"

        End If

    End Sub

End Class
```

Programa 29. Formulario con un botón que genera un número aleatorio
Controles:

- Button → Name: `btnAleatorio`, Text: "Generar"
- Label → Name: `lblNumero`, Text: ""

Código:

```
Public Class Form1

    Dim rnd As New Random()


    Private Sub btnAleatorio_Click(sender As Object, e As EventArgs) Handles
btnAleatorio.Click

        lblNumero.Text = "Número: " & rnd.Next(1, 101).ToString()

    End Sub

End Class
```

Programa 30. Formulario con un temporizador que actualiza la hora
Controles:

- Label → Name: `lblHora`, Text: "Hora: "
- Timer → Name: `Timer1`, Enabled: True, Interval: 1000

Código:

```
Public Class Form1

    Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick

        lblHora.Text = "Hora: " & DateTime.Now.ToLongTimeString()

    End Sub

End Class
```

Programa 31. Formulario con un botón que oculta y muestra una etiqueta
Controles:

- Label → Name: `lblTexto`, Text: "Soy una etiqueta"
- Button → Name: `btnOcultarMostrar`, Text: "Ocultar/Mostrar"

Código:

```
Public Class Form1

    Private Sub btnOcultarMostrar_Click(sender As Object, e As EventArgs)
Handles btnOcultarMostrar.Click

        lblTexto.Visible = Not lblTexto.Visible

    End Sub

End Class
```

Programa 32. Formulario con un botón que abre un nuevo formulario vacío
Controles:

- Button → Name: `btnAbrir`, Text: "Abrir Formulario"

Código:

```
Public Class Form1

    Private Sub btnAbrir_Click(sender As Object, e As EventArgs) Handles
btnAbrir.Click

        Dim nuevoForm As New Form

        nuevoForm.Text = "Nuevo Formulario"

        nuevoForm.Show()

    End Sub

End Class
```

Programa 33. Formulario con una barra de progreso que avanza al presionar un botón
Controles:

- ProgressBar → Name: `prgBarra`, Minimum: 0, Maximum: 100, Value: 0
- Button → Name: `btnAvanzar`, Text: "Avanzar"

Código:

```
Public Class Form1

    Private Sub btnAvanzar_Click(sender As Object, e As EventArgs) Handles
btnAvanzar.Click

        If prgBarra.Value < prgBarra.Maximum Then

            prgBarra.Value += 10

        End If

    End Sub

End Class
```

Programa 34. Formulario con un botón que copia texto al portapapeles
Controles:

- TextBox → Name: `txtEntrada`
- Button → Name: `btnCopiar`, Text: "Copiar"

Código:

```
Public Class Form1

    Private Sub btnCopiar_Click(sender As Object, e As EventArgs) Handles
btnCopiar.Click

        Clipboard.SetText(txtEntrada.Text)

    End Sub

End Class
```

Programa 35. Formulario con un cuadro de texto que limita caracteres
Controles:

- TextBox → Name: `txtLimitado`, MaxLength: 10
- Label → Name: `lblAviso`, Text: "Máximo 10 caracteres"

Código:

```
Public Class Form1

End Class
```

(Nota: El límite se aplica directamente con la propiedad `MaxLength` del TextBox, no se necesita código extra.)

Programa 36. Formulario con un botón que abre un cuadro de diálogo de color
Controles:

- Button → Name: `btnColor`, Text: "Elegir Color"
- Label → Name: `lblTexto`, Text: "Texto de ejemplo"
- ColorDialog → Name: `ColorDialog1`

Código:

```
Public Class Form1

    Private Sub btnColor_Click(sender As Object, e As EventArgs) Handles
btnColor.Click

        If ColorDialog1.ShowDialog() = DialogResult.OK Then

            lblTexto.ForeColor = ColorDialog1.Color

        End If

    End Sub

End Class
```

Programa 37. Formulario con un botón que abre un cuadro de diálogo de fuente
Controles:

- Button → Name: `btnFuente`, Text: "Elegir Fuente"
- Label → Name: `lblTexto`, Text: "Texto de ejemplo"
- FontDialog → Name: `FontDialog1`

Código:

```
Public Class Form1

    Private Sub btnFuente_Click(sender As Object, e As EventArgs) Handles
btnFuente.Click

        If FontDialog1.ShowDialog() = DialogResult.OK Then

            lblTexto.Font = FontDialog1.Font

        End If

    End Sub

End Class
```

Programa 38. Formulario con un botón que abre un cuadro de diálogo de archivo
Controles:

- Button → Name: `btnArchivo`, Text: "Abrir Archivo"
- OpenFileDialog → Name: `OpenFileDialog1`

Código:

```
Public Class Form1

    Private Sub btnArchivo_Click(sender As Object, e As EventArgs) Handles
btnArchivo.Click

        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then

            MessageBox.Show("Archivo seleccionado: " & OpenFileDialog1.FileName)

        End If

    End Sub

End Class
```

Programa 39. Formulario con un botón que abre un cuadro de diálogo de carpeta
Controles:

- Button → Name: `btnCarpeta`, Text: "Elegir Carpeta"
- FolderBrowserDialog → Name: `FolderBrowserDialog1`

Código:

```
Public Class Form1

    Private Sub btnCarpeta_Click(sender As Object, e As EventArgs) Handles
btnCarpeta.Click

        If FolderBrowserDialog1.ShowDialog() = DialogResult.OK Then

            MessageBox.Show("Carpeta seleccionada: " &
FolderBrowserDialog1.SelectedPath)

        End If

    End Sub

End Class
```

Programa 40. Formulario con un menú simple con dos opciones
Controles:

- MenuStrip → Name: `MenuStrip1`
    - ToolStripMenuItem → Name: `opcion1ToolStripMenuItem`, Text: "Opción 1"
    - ToolStripMenuItem → Name: `opcion2ToolStripMenuItem`, Text: "Opción 2"

Código:

```
Public Class Form1

    Private Sub opcion1ToolStripMenuItem_Click(sender As Object, e As EventArgs)
Handles opcion1ToolStripMenuItem.Click

        MessageBox.Show("Elegiste la Opción 1")

    End Sub


    Private Sub opcion2ToolStripMenuItem_Click(sender As Object, e As EventArgs)
Handles opcion2ToolStripMenuItem.Click

        MessageBox.Show("Elegiste la Opción 2")

    End Sub

End Class
```

Programa 41. Formulario con un menú que cambia el color de fondo
Controles:

- MenuStrip → Name: `MenuStrip1`
    - ToolStripMenuItem → Name: `rojoToolStripMenuItem`, Text: "Rojo"
    - ToolStripMenuItem → Name: `verdeToolStripMenuItem`, Text: "Verde"
    - ToolStripMenuItem → Name: `azulToolStripMenuItem`, Text: "Azul"

Código:

```
Public Class Form1

    Private Sub rojoToolStripMenuItem_Click(sender As Object, e As EventArgs)
Handles rojoToolStripMenuItem.Click

        Me.BackColor = Color.Red

    End Sub


    Private Sub verdeToolStripMenuItem_Click(sender As Object, e As EventArgs)
Handles verdeToolStripMenuItem.Click

        Me.BackColor = Color.Green

    End Sub


    Private Sub azulToolStripMenuItem_Click(sender As Object, e As EventArgs)
Handles azulToolStripMenuItem.Click

        Me.BackColor = Color.Blue

    End Sub

End Class
```

Programa 42. Formulario con un menú que muestra un mensaje
Controles:

- MenuStrip → Name: `MenuStrip1`
    - ToolStripMenuItem → Name: `mensajeToolStripMenuItem`, Text: "Mostrar Mensaje"

Código:

```
Public Class Form1

    Private Sub mensajeToolStripMenuItem_Click(sender As Object, e As EventArgs)
Handles mensajeToolStripMenuItem.Click

        MessageBox.Show("Este es un mensaje desde el menú")

    End Sub

End Class
```

Programa 43. Formulario con un menú para abrir otro formulario
Controles:

- MenuStrip → Name: `MenuStrip1`
    - ToolStripMenuItem → Name: `abrirFormToolStripMenuItem`, Text: "Abrir Formulario"

Código:

```
Public Class Form1

    Private Sub abrirFormToolStripMenuItem_Click(sender As Object, e As
EventArgs) Handles abrirFormToolStripMenuItem.Click

        Dim nuevoForm As New Form

        nuevoForm.Text = "Nuevo Formulario"

        nuevoForm.Show()

    End Sub

End Class
```

Programa 44. Formulario con una barra de herramientas (ToolStrip)
Controles:

- ToolStrip → Name: `ToolStrip1`
    - ToolStripButton → Name: `btnNuevo`, Text: "Nuevo"
    - ToolStripButton → Name: `btnSalir`, Text: "Salir"

Código:

```
Public Class Form1

    Private Sub btnNuevo_Click(sender As Object, e As EventArgs) Handles
btnNuevo.Click

        MessageBox.Show("Has pulsado 'Nuevo'")

    End Sub


    Private Sub btnSalir_Click(sender As Object, e As EventArgs) Handles
btnSalir.Click

        Application.Exit()

    End Sub

End Class
```

Programa 45. Formulario con un cuadro de texto que busca en un ListBox
Controles:

- TextBox → Name: `txtBuscar`
- Button → Name: `btnBuscar`, Text: "Buscar"
- ListBox → Name: `lstElementos`

Código:

```
Public Class Form1

    Private Sub btnBuscar_Click(sender As Object, e As EventArgs) Handles
btnBuscar.Click

        Dim item As String = txtBuscar.Text

        Dim index As Integer = lstElementos.FindString(item)

        If index <> -1 Then

            lstElementos.SelectedIndex = index

        Else

            MessageBox.Show("Elemento no encontrado")

        End If

    End Sub

End Class
```

Programa 46. Formulario con un botón que cambia el título del formulario
Controles:

- Button → Name: `btnTitulo`, Text: "Cambiar Título"

Código:

```
Public Class Form1

    Private Sub btnTitulo_Click(sender As Object, e As EventArgs) Handles
btnTitulo.Click

        Me.Text = "Nuevo Título del Formulario"

    End Sub

End Class
```

Programa 47. Formulario con dos botones que mueven un Label en el formulario
Controles:

- Label → Name: `lblMover`, Text: "Mueve este texto"
- Button → Name: `btnIzquierda`, Text: "Izquierda"
- Button → Name: `btnDerecha`, Text: "Derecha"

Código:

```
Public Class Form1

    Private Sub btnIzquierda_Click(sender As Object, e As EventArgs) Handles
btnIzquierda.Click

        lblMover.Left -= 10

    End Sub


    Private Sub btnDerecha_Click(sender As Object, e As EventArgs) Handles
btnDerecha.Click

        lblMover.Left += 10

    End Sub

End Class
```

Programa 48. Formulario con un botón que muestra la fecha actual
Controles:

- Button → Name: `btnFecha`, Text: "Mostrar Fecha"
- Label → Name: `lblFecha`, Text: ""

Código:

```
Public Class Form1

    Private Sub btnFecha_Click(sender As Object, e As EventArgs) Handles
btnFecha.Click

        lblFecha.Text = "Hoy es: " & DateTime.Now.ToLongDateString()

    End Sub

End Class
```

Programa 49. Formulario con un PictureBox que muestra diferentes imágenes
Controles:

- PictureBox → Name: `picImagen`, SizeMode: StretchImage, Size: 200x200
- Button → Name: `btnImagen1`, Text: "Imagen 1"
- Button → Name: `btnImagen2`, Text: "Imagen 2"

Código:

```
Public Class Form1

    Private Sub btnImagen1_Click(sender As Object, e As EventArgs) Handles
btnImagen1.Click

        picImagen.Image = Image.FromFile("imagen1.jpg")

    End Sub


    Private Sub btnImagen2_Click(sender As Object, e As EventArgs) Handles
btnImagen2.Click

        picImagen.Image = Image.FromFile("imagen2.jpg")

    End Sub

End Class
```

Programa 50. Formulario con un botón que guarda texto en un archivo
Controles:

- TextBox → Name: `txtContenido`, Multiline: True, Size: 200x100
- Button → Name: `btnGuardar`, Text: "Guardar"
- SaveFileDialog → Name: `SaveFileDialog1`

Código:

```
Imports System.IO


Public Class Form1

    Private Sub btnGuardar_Click(sender As Object, e As EventArgs) Handles
btnGuardar.Click

        If SaveFileDialog1.ShowDialog() = DialogResult.OK Then

            File.WriteAllText(SaveFileDialog1.FileName, txtContenido.Text)

            MessageBox.Show("Archivo guardado con éxito")

        End If

    End Sub

End Class
```

Categoría 2: Programas intermedios (51–150)

Programa 51. Agenda de contactos básica con ListBox

Controles:

- TextBox → Name: `txtNombre`
- Button → Name: `btnAgregar`, Text: "Agregar"
- ListBox → Name: `lstContactos`

Código:

```
Public Class Form1

    Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click

        If txtNombre.Text <> "" Then

            lstContactos.Items.Add(txtNombre.Text)

            txtNombre.Clear()

        End If

    End Sub

End Class
```

Programa 52. Calculadora simple (suma, resta, multiplicación, división)
Controles:

- TextBox → Name: `txtNum1`
- TextBox → Name: `txtNum2`
- Button → Name: `btnSumar`, Text: "Sumar"
- Button → Name: `btnRestar`, Text: "Restar"
- Button → Name: `btnMultiplicar`, Text: "Multiplicar"
- Button → Name: `btnDividir`, Text: "Dividir"
- Label → Name: `lblResultado`

Código:

```
Public Class Form1

    Private Sub btnSumar_Click(sender As Object, e As EventArgs) Handles
btnSumar.Click

        lblResultado.Text = (Val(txtNum1.Text) + Val(txtNum2.Text)).ToString()

    End Sub


    Private Sub btnRestar_Click(sender As Object, e As EventArgs) Handles
btnRestar.Click

        lblResultado.Text = (Val(txtNum1.Text) - Val(txtNum2.Text)).ToString()

    End Sub


    Private Sub btnMultiplicar_Click(sender As Object, e As EventArgs) Handles
btnMultiplicar.Click

        lblResultado.Text = (Val(txtNum1.Text) * Val(txtNum2.Text)).ToString()

    End Sub


    Private Sub btnDividir_Click(sender As Object, e As EventArgs) Handles
btnDividir.Click

        If Val(txtNum2.Text) <> 0 Then

            lblResultado.Text = (Val(txtNum1.Text) /
Val(txtNum2.Text)).ToString()

        Else

            lblResultado.Text = "Error: división por 0"

        End If

    End Sub

End Class
```

Programa 53. Conversor de temperatura (Celsius-Fahrenheit)
Controles:

- TextBox → Name: `txtCelsius`
- Button → Name: `btnConvertir`, Text: "Convertir"
- Label → Name: `lblResultado`

Código:

```
Public Class Form1

    Private Sub btnConvertir_Click(sender As Object, e As EventArgs) Handles
btnConvertir.Click

        Dim c As Double = Val(txtCelsius.Text)

        Dim f As Double = (c * 9 / 5) + 32

        lblResultado.Text = c & " °C = " & f & " °F"

    End Sub

End Class
```

Programa 54. Conversor de divisas
Controles:

- TextBox → Name: `txtEuros`
- Button → Name: `btnConvertir`, Text: "Convertir a USD"
- Label → Name: `lblResultado`

Código:

```
Public Class Form1

    Private Sub btnConvertir_Click(sender As Object, e As EventArgs) Handles
btnConvertir.Click

        Dim euros As Double = Val(txtEuros.Text)

        Dim tasa As Double = 1.1 ' Ejemplo de tasa EUR a USD

        lblResultado.Text = euros & " € = " & (euros * tasa) & " $"

    End Sub

End Class
```

Programa 55. Bloc de notas básico
Controles:

- TextBox → Name: `txtContenido`, Multiline: True, Dock: Fill

Código:

```
Public Class Form1

    ' Sin eventos, solo un TextBox grande para escribir

End Class
```

Programa 56. Aplicación que guarda y carga texto desde archivo
Controles:

- TextBox → Name: `txtContenido`, Multiline: True, Size: 300x200
- Button → Name: `btnGuardar`, Text: "Guardar"
- Button → Name: `btnCargar`, Text: "Cargar"
- OpenFileDialog → Name: `OpenFileDialog1`
- SaveFileDialog → Name: `SaveFileDialog1`

Código:

```
Imports System.IO


Public Class Form1

    Private Sub btnGuardar_Click(sender As Object, e As EventArgs) Handles
btnGuardar.Click

        If SaveFileDialog1.ShowDialog() = DialogResult.OK Then

            File.WriteAllText(SaveFileDialog1.FileName, txtContenido.Text)

        End If

    End Sub


    Private Sub btnCargar_Click(sender As Object, e As EventArgs) Handles
btnCargar.Click

        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then

            txtContenido.Text = File.ReadAllText(OpenFileDialog1.FileName)

        End If

    End Sub

End Class
```

Programa 57. Aplicación que cuenta palabras en un TextBox
Controles:

- TextBox → Name: `txtTexto`, Multiline: True, Size: 300x100
- Button → Name: `btnContar`, Text: "Contar Palabras"
- Label → Name: `lblResultado`

Código:

```
Public Class Form1

    Private Sub btnContar_Click(sender As Object, e As EventArgs) Handles
btnContar.Click

        Dim palabras() As String = txtTexto.Text.Split({" "c, vbCrLf},
StringSplitOptions.RemoveEmptyEntries)

        lblResultado.Text = "Número de palabras: " & palabras.Length

    End Sub

End Class
```

Programa 58. Aplicación que muestra tablas de multiplicar
Controles:

- TextBox → Name: `txtNumero`
- Button → Name: `btnGenerar`, Text: "Generar Tabla"
- ListBox → Name: `lstTabla`

Código:

```
Public Class Form1

    Private Sub btnGenerar_Click(sender As Object, e As EventArgs) Handles
btnGenerar.Click

        lstTabla.Items.Clear()

        Dim n As Integer = Val(txtNumero.Text)

        For i As Integer = 1 To 10

            lstTabla.Items.Add(n & " x " & i & " = " & (n * i))

        Next

    End Sub

End Class
```

Programa 59. Aplicación que genera contraseñas aleatorias
Controles:

- Button → Name: `btnGenerar`, Text: "Generar Contraseña"
- Label → Name: `lblPassword`

Código:

```
Imports System.Text


Public Class Form1

    Private Sub btnGenerar_Click(sender As Object, e As EventArgs) Handles
btnGenerar.Click

        Dim chars As String =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@#$%"

        Dim rnd As New Random()

        Dim sb As New StringBuilder()

        For i As Integer = 1 To 8

            sb.Append(chars(rnd.Next(chars.Length)))

        Next

        lblPassword.Text = sb.ToString()

    End Sub

End Class
```

Programa 60. Aplicación que muestra mensajes aleatorios
Controles:

- Button → Name: `btnMensaje`, Text: "Mostrar Mensaje"
- Label → Name: `lblMensaje`

Código:

```
Public Class Form1

    Dim mensajes() As String = {"Hola!", "Bienvenido", "VB.NET es genial",
"Sigue aprendiendo", "Suerte con tus proyectos"}


    Private Sub btnMensaje_Click(sender As Object, e As EventArgs) Handles
btnMensaje.Click

        Dim rnd As New Random()

        Dim index As Integer = rnd.Next(mensajes.Length)

        lblMensaje.Text = mensajes(index)

    End Sub

End Class
```

Programa 61. Aplicación que calcula el IMC
Controles:

- TextBox → Name: `txtPeso`, Placeholder: "Peso (kg)"
- TextBox → Name: `txtAltura`, Placeholder: "Altura (m)"
- Button → Name: `btnCalcular`, Text: "Calcular IMC"
- Label → Name: `lblResultado`

Código:

```
Public Class Form1

    Private Sub btnCalcular_Click(sender As Object, e As EventArgs) Handles
btnCalcular.Click

        Dim peso As Double = Val(txtPeso.Text)

        Dim altura As Double = Val(txtAltura.Text)

        If altura > 0 Then

            Dim imc As Double = peso / (altura * altura)

            lblResultado.Text = "IMC: " & imc.ToString("0.00")

        Else

            lblResultado.Text = "Altura inválida"

        End If

    End Sub

End Class
```

Programa 62. Conversor de unidades de longitud (metros a cm, mm, km)
Controles:

- TextBox → Name: `txtMetros`
- Button → Name: `btnConvertir`, Text: "Convertir"
- ListBox → Name: `lstResultados`

Código:

```
Public Class Form1

    Private Sub btnConvertir_Click(sender As Object, e As EventArgs) Handles
btnConvertir.Click

        lstResultados.Items.Clear()

        Dim m As Double = Val(txtMetros.Text)

        lstResultados.Items.Add(m & " m = " & (m * 100) & " cm")

        lstResultados.Items.Add(m & " m = " & (m * 1000) & " mm")

        lstResultados.Items.Add(m & " m = " & (m / 1000) & " km")

    End Sub

End Class
```

Programa 63. Conversor de unidades de peso (kg a g, lb)
Controles:

- TextBox → Name: `txtKg`
- Button → Name: `btnConvertir`, Text: "Convertir"
- ListBox → Name: `lstResultados`

Código:

```
Public Class Form1

    Private Sub btnConvertir_Click(sender As Object, e As EventArgs) Handles
btnConvertir.Click

        lstResultados.Items.Clear()

        Dim kg As Double = Val(txtKg.Text)

        lstResultados.Items.Add(kg & " kg = " & (kg * 1000) & " g")

        lstResultados.Items.Add(kg & " kg = " & (kg * 2.20462) & " lb")

    End Sub

End Class
```

Programa 64. Conversor de unidades de tiempo (horas a minutos y segundos)
Controles:

- TextBox → Name: `txtHoras`
- Button → Name: `btnConvertir`, Text: "Convertir"
- Label → Name: `lblResultado`

Código:

```
Public Class Form1

    Private Sub btnConvertir_Click(sender As Object, e As EventArgs) Handles
btnConvertir.Click

        Dim h As Double = Val(txtHoras.Text)

        lblResultado.Text = h & " h = " & (h * 60) & " min = " & (h * 3600) & "
seg"

    End Sub

End Class
```

Programa 65. Conversor de unidades de velocidad (km/h a m/s)
Controles:

- TextBox → Name: `txtKmH`
- Button → Name: `btnConvertir`, Text: "Convertir"
- Label → Name: `lblResultado`

Código:

```
Public Class Form1

    Private Sub btnConvertir_Click(sender As Object, e As EventArgs) Handles
btnConvertir.Click

        Dim kmh As Double = Val(txtKmH.Text)

        Dim ms As Double = kmh / 3.6

        lblResultado.Text = kmh & " km/h = " & ms.ToString("0.00") & " m/s"

    End Sub

End Class
```

Programa 66. Aplicación que guarda configuraciones en My.Settings
Controles:

- TextBox → Name: `txtDato`
- Button → Name: `btnGuardar`, Text: "Guardar"
- Button → Name: `btnCargar`, Text: "Cargar"
- Label → Name: `lblResultado`

Código:

```
Public Class Form1

    Private Sub btnGuardar_Click(sender As Object, e As EventArgs) Handles
btnGuardar.Click

        My.Settings.MiDato = txtDato.Text

        My.Settings.Save()

    End Sub


    Private Sub btnCargar_Click(sender As Object, e As EventArgs) Handles
btnCargar.Click

        lblResultado.Text = "Dato guardado: " & My.Settings.MiDato

    End Sub

End Class
```

(En My Project > Settings, agregar `MiDato` como tipo String, alcance User.)

Programa 67. Calculadora de área de figuras geométricas (círculo y cuadrado)
Controles:

- TextBox → Name: `txtRadio` (para círculo)
- TextBox → Name: `txtLado` (para cuadrado)
- Button → Name: `btnCalcular`, Text: "Calcular Áreas"
- Label → Name: `lblResultado`

Código:

```
Public Class Form1

    Private Sub btnCalcular_Click(sender As Object, e As EventArgs) Handles
btnCalcular.Click

        Dim radio As Double = Val(txtRadio.Text)

        Dim lado As Double = Val(txtLado.Text)

        Dim areaCirculo As Double = Math.PI * radio ^ 2

        Dim areaCuadrado As Double = lado * lado

        lblResultado.Text = "Área círculo: " & areaCirculo.ToString("0.00") &
vbCrLf &

                            "Área cuadrado: " & areaCuadrado

    End Sub

End Class
```

Programa 68. Calculadora de perímetro de figuras geométricas (círculo y cuadrado)
Controles:

- TextBox → Name: `txtRadio`
- TextBox → Name: `txtLado`
- Button → Name: `btnCalcular`, Text: "Calcular Perímetros"
- Label → Name: `lblResultado`

Código:

```
Public Class Form1

    Private Sub btnCalcular_Click(sender As Object, e As EventArgs) Handles
btnCalcular.Click

        Dim radio As Double = Val(txtRadio.Text)

        Dim lado As Double = Val(txtLado.Text)

        Dim perCirculo As Double = 2 * Math.PI * radio

        Dim perCuadrado As Double = 4 * lado

        lblResultado.Text = "Perímetro círculo: " & perCirculo.ToString("0.00")
& vbCrLf &

                            "Perímetro cuadrado: " & perCuadrado

    End Sub

End Class
```

Programa 69. Juego del número secreto
Controles:

- TextBox → Name: `txtIntento`
- Button → Name: `btnProbar`, Text: "Probar"
- Label → Name: `lblResultado`

Código:

```
Public Class Form1
    Dim secreto As Integer
    Dim rnd As New Random()

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        secreto = rnd.Next(1, 101) ' Número entre 1 y 100
    End Sub

    Private Sub btnProbar_Click(sender As Object, e As EventArgs) Handles
btnProbar.Click
        Dim intento As Integer = Val(txtIntento.Text)
        If intento = secreto Then
            lblResultado.Text = "¡Correcto!"
        ElseIf intento < secreto Then
            lblResultado.Text = "El número es mayor"
        Else
            lblResultado.Text = "El número es menor"
        End If
    End Sub
End Class
```

Programa 70. Juego de adivinar palabra
Controles:

- Label → Name: `lblPista`, Text: "Palabra: _ _ _ _"
- TextBox → Name: `txtLetra`
- Button → Name: `btnProbar`, Text: "Probar Letra"
- Label → Name: `lblResultado`

Código:

```
Public Class Form1
    Dim palabra As String = "GATO"
    Dim adivinada As String = "____"

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        lblPista.Text = "Palabra: " & adivinada
    End Sub

    Private Sub btnProbar_Click(sender As Object, e As EventArgs) Handles
btnProbar.Click
        Dim letra As String = txtLetra.Text.ToUpper()
        If letra.Length = 1 Then
            Dim nueva As Char() = adivinada.ToCharArray()
            For i As Integer = 0 To palabra.Length - 1
                If palabra(i) = letra(0) Then
                    nueva(i) = letra(0)
                End If
            Next
            adivinada = New String(nueva)
            lblPista.Text = "Palabra: " & adivinada
            If adivinada = palabra Then
                lblResultado.Text = "¡Ganaste!"
            End If
        End If
        txtLetra.Clear()
    End Sub
End Class
```

Programa 71. Juego de piedra, papel o tijera
Controles:

- ComboBox → Name: `cmbJugador`, Items: {"Piedra", "Papel", "Tijera"}
- Button → Name: `btnJugar`, Text: "Jugar"
- Label → Name: `lblResultado`

Código:

```
Public Class Form1
    Dim opciones() As String = {"Piedra", "Papel", "Tijera"}
    Dim rnd As New Random()

    Private Sub btnJugar_Click(sender As Object, e As EventArgs) Handles
btnJugar.Click
        Dim jugador As String = cmbJugador.Text
        Dim pc As String = opciones(rnd.Next(opciones.Length))
        If jugador = "" Then
            lblResultado.Text = "Elige una opción"
            Exit Sub
        End If

        If jugador = pc Then
            lblResultado.Text = "Empate, PC eligió " & pc
        ElseIf (jugador = "Piedra" And pc = "Tijera") Or
               (jugador = "Papel" And pc = "Piedra") Or
               (jugador = "Tijera" And pc = "Papel") Then
            lblResultado.Text = "Ganaste, PC eligió " & pc
        Else
            lblResultado.Text = "Perdiste, PC eligió " & pc
        End If
    End Sub
End Class
```

Programa 72. Cronómetro con Start, Stop y Reset
Controles:

- Label → Name: `lblTiempo`, Text: "00:00:00"
- Button → Name: `btnStart`, Text: "Start"
- Button → Name: `btnStop`, Text: "Stop"
- Button → Name: `btnReset`, Text: "Reset"
- Timer → Name: `Timer1`, Interval: 1000

Código:

```
Public Class Form1

    Dim segundos As Integer = 0


    Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick

        segundos += 1

        lblTiempo.Text = TimeSpan.FromSeconds(segundos).ToString("hh\:mm\:ss")

    End Sub


    Private Sub btnStart_Click(sender As Object, e As EventArgs) Handles
btnStart.Click

        Timer1.Start()

    End Sub


    Private Sub btnStop_Click(sender As Object, e As EventArgs) Handles
btnStop.Click

        Timer1.Stop()

    End Sub


    Private Sub btnReset_Click(sender As Object, e As EventArgs) Handles
btnReset.Click

        Timer1.Stop()

        segundos = 0

        lblTiempo.Text = "00:00:00"

    End Sub

End Class
```

Programa 73. Temporizador con alarma
Controles:

- TextBox → Name: `txtSegundos`, Placeholder: "Segundos"
- Button → Name: `btnIniciar`, Text: "Iniciar"
- Label → Name: `lblTiempo`
- Timer → Name: `Timer1`, Interval: 1000

Código:

```
Public Class Form1
    Dim tiempo As Integer

    Private Sub btnIniciar_Click(sender As Object, e As EventArgs) Handles
btnIniciar.Click
        tiempo = Val(txtSegundos.Text)
        Timer1.Start()
    End Sub

    Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
        If tiempo > 0 Then
            tiempo -= 1
            lblTiempo.Text = tiempo & " s"
        Else
            Timer1.Stop()
            MessageBox.Show("¡Tiempo terminado!")
        End If
    End Sub
End Class
```

Programa 74. Editor de listas (agregar, modificar y borrar)
Controles:

- TextBox → Name: `txtElemento`
- Button → Name: `btnAgregar`, Text: "Agregar"
- Button → Name: `btnModificar`, Text: "Modificar"
- Button → Name: `btnBorrar`, Text: "Borrar"
- ListBox → Name: `lstElementos`

Código:

```
Public Class Form1

    Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click

        lstElementos.Items.Add(txtElemento.Text)

        txtElemento.Clear()

    End Sub


    Private Sub btnModificar_Click(sender As Object, e As EventArgs) Handles
btnModificar.Click

        If lstElementos.SelectedIndex >= 0 Then

            lstElementos.Items(lstElementos.SelectedIndex) = txtElemento.Text

        End If

    End Sub


    Private Sub btnBorrar_Click(sender As Object, e As EventArgs) Handles
btnBorrar.Click

        If lstElementos.SelectedIndex >= 0 Then

            lstElementos.Items.RemoveAt(lstElementos.SelectedIndex)

        End If

    End Sub

End Class
```

Programa 75. Editor de texto con guardar y abrir archivo
Controles:

- TextBox → Name: `txtContenido`, Multiline: True, Dock: Fill
- MenuStrip → Items: {"Archivo → Abrir", "Archivo → Guardar"}
- OpenFileDialog → Name: `OpenFileDialog1`
- SaveFileDialog → Name: `SaveFileDialog1`

Código:

```
Imports System.IO


Public Class Form1

    Private Sub AbrirToolStripMenuItem_Click(sender As Object, e As EventArgs)
Handles AbrirToolStripMenuItem.Click

        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then

            txtContenido.Text = File.ReadAllText(OpenFileDialog1.FileName)

        End If

    End Sub


    Private Sub GuardarToolStripMenuItem_Click(sender As Object, e As EventArgs)
Handles GuardarToolStripMenuItem.Click

        If SaveFileDialog1.ShowDialog() = DialogResult.OK Then

            File.WriteAllText(SaveFileDialog1.FileName, txtContenido.Text)

        End If

    End Sub

End Class
```

Programa 76. Aplicación con varios formularios enlazados
Controles (Form1):

- Button → Name: `btnAbrir`, Text: "Abrir Form2"

Form2: vacío con un Label.

Código:

```
' Form1

Public Class Form1

    Private Sub btnAbrir_Click(sender As Object, e As EventArgs) Handles
btnAbrir.Click

        Dim f2 As New Form2

        f2.Show()

    End Sub

End Class


' Form2

Public Class Form2

End Class
```

Programa 77. Agenda que guarda datos en archivo de texto
Controles:

- TextBox → Name: `txtNombre`
- TextBox → Name: `txtTelefono`
- Button → Name: `btnGuardar`, Text: "Guardar"
- ListBox → Name: `lstContactos`

Código:

```
Imports System.IO


Public Class Form1

    Private Sub btnGuardar_Click(sender As Object, e As EventArgs) Handles
btnGuardar.Click

        Dim contacto As String = txtNombre.Text & " - " & txtTelefono.Text

        File.AppendAllText("agenda.txt", contacto & Environment.NewLine)

        lstContactos.Items.Add(contacto)

        txtNombre.Clear()

        txtTelefono.Clear()

    End Sub

End Class
```

Programa 78. Agenda que guarda datos en archivo CSV
Controles:

- TextBox → Name: `txtNombre`
- TextBox → Name: `txtTelefono`
- Button → Name: `btnGuardar`, Text: "Guardar"
- ListBox → Name: `lstContactos`

Código:

```
Imports System.IO


Public Class Form1

    Private Sub btnGuardar_Click(sender As Object, e As EventArgs) Handles
btnGuardar.Click

        Dim contacto As String = txtNombre.Text & "," & txtTelefono.Text

        File.AppendAllText("agenda.csv", contacto & Environment.NewLine)

        lstContactos.Items.Add(contacto)

        txtNombre.Clear()

        txtTelefono.Clear()

    End Sub

End Class
```

Programa 79. Agenda que guarda datos en archivo XML
Controles:

- TextBox → Name: `txtNombre`
- TextBox → Name: `txtTelefono`
- Button → Name: `btnGuardar`, Text: "Guardar"
- ListBox → Name: `lstContactos`

Código:

```
Imports System.Xml


Public Class Form1

    Private Sub btnGuardar_Click(sender As Object, e As EventArgs) Handles
btnGuardar.Click

        Dim doc As New XmlDocument()

        Dim archivo As String = "agenda.xml"


        If IO.File.Exists(archivo) Then

            doc.Load(archivo)

        Else

            Dim root As XmlElement = doc.CreateElement("Contactos")

            doc.AppendChild(root)

        End If


        Dim contacto As XmlElement = doc.CreateElement("Contacto")

        contacto.SetAttribute("Nombre", txtNombre.Text)

        contacto.SetAttribute("Telefono", txtTelefono.Text)

        doc.DocumentElement.AppendChild(contacto)

        doc.Save(archivo)


        lstContactos.Items.Add(txtNombre.Text & " - " & txtTelefono.Text)

        txtNombre.Clear()

        txtTelefono.Clear()

    End Sub

End Class
```

Programa 80. Agenda que guarda datos en base de datos Access
Controles:

- TextBox → Name: `txtNombre`
- TextBox → Name: `txtTelefono`
- Button → Name: `btnGuardar`, Text: "Guardar"
- ListBox → Name: `lstContactos`

Código:

```
Imports System.Data.OleDb


Public Class Form1

    Dim conexion As New OleDbConnection("Provider=Microsoft.ACE.OLEDB.12.0;Data Source=agenda.accdb;")


    Private Sub btnGuardar_Click(sender As Object, e As EventArgs) Handles btnGuardar.Click

        Dim cmd As New OleDbCommand("INSERT INTO Contactos (Nombre, Telefono) VALUES (@n, @t)", conexion)

        cmd.Parameters.AddWithValue("@n", txtNombre.Text)

        cmd.Parameters.AddWithValue("@t", txtTelefono.Text)

        conexion.Open()

        cmd.ExecuteNonQuery()

        conexion.Close()


        lstContactos.Items.Add(txtNombre.Text & " - " & txtTelefono.Text)

        txtNombre.Clear()

        txtTelefono.Clear()

    End Sub

End Class
```

(Requiere tener una base de datos Access `agenda.accdb` con tabla `Contactos(Nombre, Telefono)`.)

Programa 81. Formulario con gráficos simples
Controles:

- Button → Name: `btnDibujar`, Text: "Dibujar"
- PictureBox → Name: `picGrafico`, Size: 300x200, BorderStyle: FixedSingle

Código:

```
Public Class Form1

    Private Sub btnDibujar_Click(sender As Object, e As EventArgs) Handles
btnDibujar.Click

        Dim g As Graphics = picGrafico.CreateGraphics()

        g.Clear(Color.White)

        g.DrawLine(Pens.Blue, 10, 10, 200, 100)

        g.DrawRectangle(Pens.Red, 50, 50, 100, 50)

        g.DrawEllipse(Pens.Green, 100, 30, 80, 40)

    End Sub

End Class
```

Programa 82. Formulario con impresión de texto
Controles:

- TextBox → Name: `txtTexto`, Multiline: True, Size: 300x150
- Button → Name: `btnImprimir`, Text: "Imprimir"
- PrintDocument → Name: `PrintDocument1`
- PrintDialog → Name: `PrintDialog1`

Código:

```
Public Class Form1

    Private Sub btnImprimir_Click(sender As Object, e As EventArgs) Handles
btnImprimir.Click

        If PrintDialog1.ShowDialog() = DialogResult.OK Then

            PrintDocument1.Print()

        End If

    End Sub


    Private Sub PrintDocument1_PrintPage(sender As Object, e As
Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage

        e.Graphics.DrawString(txtTexto.Text, New Font("Arial", 12),
Brushes.Black, 50, 50)

    End Sub

End Class
```

Programa 83. Calculadora de ecuaciones de segundo grado
Controles:

- TextBox → Name: `txtA`
- TextBox → Name: `txtB`
- TextBox → Name: `txtC`
- Button → Name: `btnCalcular`, Text: "Calcular"
- Label → Name: `lblResultado`

Código:

```
Public Class Form1

    Private Sub btnCalcular_Click(sender As Object, e As EventArgs) Handles
btnCalcular.Click

        Dim a As Double = Val(txtA.Text)

        Dim b As Double = Val(txtB.Text)

        Dim c As Double = Val(txtC.Text)

        Dim d As Double = b ^ 2 - 4 * a * c

        If d < 0 Then

            lblResultado.Text = "No hay soluciones reales"

        Else

            Dim x1 As Double = (-b + Math.Sqrt(d)) / (2 * a)

            Dim x2 As Double = (-b - Math.Sqrt(d)) / (2 * a)

            lblResultado.Text = "x1=" & x1 & ", x2=" & x2

        End If

    End Sub

End Class
```

Programa 84. Calculadora de interés simple
Controles:

- TextBox → Name: `txtCapital`
- TextBox → Name: `txtInteres` (porcentaje)
- TextBox → Name: `txtTiempo` (años)
- Button → Name: `btnCalcular`, Text: "Calcular"
- Label → Name: `lblResultado`

Código:

```
Public Class Form1

    Private Sub btnCalcular_Click(sender As Object, e As EventArgs) Handles
btnCalcular.Click

        Dim C As Double = Val(txtCapital.Text)

        Dim i As Double = Val(txtInteres.Text) / 100

        Dim t As Double = Val(txtTiempo.Text)

        Dim interés As Double = C * i * t

        lblResultado.Text = "Interés simple: " & interés

    End Sub

End Class
```

Programa 85. Calculadora de interés compuesto
Controles:

- TextBox → Name: `txtCapital`
- TextBox → Name: `txtInteres` (porcentaje)
- TextBox → Name: `txtTiempo` (años)
- Button → Name: `btnCalcular`, Text: "Calcular"
- Label → Name: `lblResultado`

Código:

```
Public Class Form1

    Private Sub btnCalcular_Click(sender As Object, e As EventArgs) Handles
btnCalcular.Click

        Dim C As Double = Val(txtCapital.Text)

        Dim i As Double = Val(txtInteres.Text) / 100

        Dim t As Double = Val(txtTiempo.Text)

        Dim monto As Double = C * (1 + i) ^ t

        lblResultado.Text = "Monto con interés compuesto: " &
monto.ToString("0.00")

    End Sub

End Class
```

Programa 86. Conversor de números decimales a binarios
Controles:

- TextBox → Name: `txtDecimal`
- Button → Name: `btnConvertir`, Text: "Convertir"
- Label → Name: `lblBinario`

Código:

```
Public Class Form1

    Private Sub btnConvertir_Click(sender As Object, e As EventArgs) Handles
btnConvertir.Click

        Dim n As Integer = Val(txtDecimal.Text)

        lblBinario.Text = "Binario: " & Convert.ToString(n, 2)

    End Sub

End Class
```

Programa 87. Conversor de binarios a decimales
Controles:

- TextBox → Name: `txtBinario`
- Button → Name: `btnConvertir`, Text: "Convertir"
- Label → Name: `lblDecimal`

Código:

```
Public Class Form1

    Private Sub btnConvertir_Click(sender As Object, e As EventArgs) Handles
btnConvertir.Click

        Dim bin As String = txtBinario.Text

        Try

            lblDecimal.Text = "Decimal: " & Convert.ToInt32(bin, 2)

        Catch ex As Exception

            lblDecimal.Text = "Número binario inválido"

        End Try

    End Sub

End Class
```

Programa 88. Conversor de números romanos a enteros
Controles:

- TextBox → Name: `txtRomano`
- Button → Name: `btnConvertir`, Text: "Convertir"
- Label → Name: `lblEntero`

Código:

```
Public Class Form1

    Private Sub btnConvertir_Click(sender As Object, e As EventArgs) Handles
btnConvertir.Click

        Dim roman As String = txtRomano.Text.ToUpper()

        Dim dic As New Dictionary(Of Char, Integer) From {

            {"I"c, 1}, {"V"c, 5}, {"X"c, 10}, {"L"c, 50},

            {"C"c, 100}, {"D"c, 500}, {"M"c, 1000}

        }

        Dim total As Integer = 0

        Dim prev As Integer = 0

        For i As Integer = roman.Length - 1 To 0 Step -1

            Dim val As Integer = dic(roman(i))

            If val < prev Then

                total -= val

            Else

                total += val

            End If

            prev = val

        Next

        lblEntero.Text = "Entero: " & total

    End Sub

End Class
```

Programa 89. Conversor de horas a minutos y segundos
Controles:

- TextBox → Name: `txtHoras`
- Button → Name: `btnConvertir`, Text: "Convertir"
- Label → Name: `lblResultado`

Código:

```
Public Class Form1

    Private Sub btnConvertir_Click(sender As Object, e As EventArgs) Handles
btnConvertir.Click

        Dim h As Double = Val(txtHoras.Text)

        lblResultado.Text = h & " horas = " & (h * 60) & " minutos = " & (h *
3600) & " segundos"

    End Sub

End Class
```

Programa 90. Conversor de kilogramos a libras
Controles:

- TextBox → Name: `txtKg`
- Button → Name: `btnConvertir`, Text: "Convertir"
- Label → Name: `lblResultado`

Código:

```
Public Class Form1

    Private Sub btnConvertir_Click(sender As Object, e As EventArgs) Handles
btnConvertir.Click

        Dim kg As Double = Val(txtKg.Text)

        Dim lb As Double = kg * 2.20462

        lblResultado.Text = kg & " kg = " & lb.ToString("0.00") & " lb"

    End Sub

End Class
```

Programa 91. Conversor de metros a pies
Controles:

- TextBox → Name: `txtMetros`
- Button → Name: `btnConvertir`, Text: "Convertir"
- Label → Name: `lblResultado`

Código:

```
Public Class Form1

    Private Sub btnConvertir_Click(sender As Object, e As EventArgs) Handles
btnConvertir.Click

        Dim metros As Double = Val(txtMetros.Text)

        Dim pies As Double = metros * 3.28084

        lblResultado.Text = metros & " m = " & pies.ToString("0.00") & " ft"

    End Sub

End Class
```

Programa 92. Conversor de Celsius a Kelvin
Controles:

- TextBox → Name: `txtCelsius`
- Button → Name: `btnConvertir`, Text: "Convertir"
- Label → Name: `lblResultado`

Código:

```
Public Class Form1

    Private Sub btnConvertir_Click(sender As Object, e As EventArgs) Handles
btnConvertir.Click

        Dim c As Double = Val(txtCelsius.Text)

        Dim k As Double = c + 273.15

        lblResultado.Text = c & " °C = " & k.ToString("0.00") & " K"

    End Sub

End Class
```

Programa 93. Conversor de binario a hexadecimal
Controles:

- TextBox → Name: `txtBinario`
- Button → Name: `btnConvertir`, Text: "Convertir"
- Label → Name: `lblHex`

Código:

```
Public Class Form1

    Private Sub btnConvertir_Click(sender As Object, e As EventArgs) Handles
btnConvertir.Click

        Try

            Dim n As Integer = Convert.ToInt32(txtBinario.Text, 2)

            lblHex.Text = "Hexadecimal: " & Hex(n)

        Catch ex As Exception

            lblHex.Text = "Número binario inválido"

        End Try

    End Sub

End Class
```

Programa 94. Conversor de hexadecimal a decimal
Controles:

- TextBox → Name: `txtHex`
- Button → Name: `btnConvertir`, Text: "Convertir"
- Label → Name: `lblDecimal`

Código:

```
Public Class Form1

    Private Sub btnConvertir_Click(sender As Object, e As EventArgs) Handles
btnConvertir.Click

        Try

            Dim n As Integer = Convert.ToInt32(txtHex.Text, 16)

            lblDecimal.Text = "Decimal: " & n

        Catch ex As Exception

            lblDecimal.Text = "Número hexadecimal inválido"

        End Try

    End Sub

End Class
```

Programa 95. Juego de memoria con botones
Controles:

- 4 Buttons → Name: `btn1`, `btn2`, `btn3`, `btn4` (colores aleatorios)

Código:

```
Public Class Form1
    Dim colores() As Color = {Color.Red, Color.Green, Color.Blue, Color.Yellow}
    Dim secuencia() As Integer = {0, 1, 2, 3}
    Dim indice As Integer = 0

    Private Sub btn_Click(sender As Object, e As EventArgs) Handles btn1.Click, btn2.Click, btn3.Click, btn4.Click
        Dim btn As Button = CType(sender, Button)
        If btn.BackColor = colores(secuencia(indice)) Then
            indice += 1
            If indice = secuencia.Length Then
                MessageBox.Show("¡Ganaste!")
                indice = 0
            End If
        Else
            MessageBox.Show("Error")
            indice = 0
        End If
    End Sub
End Class
```

(Asignar `BackColor` aleatorio a cada botón en Form_Load si se desea.)

Programa 96. Juego de laberinto simple
Controles:

- Panel → Name: `pnlLaberinto` (paredes)
- Button → Name: `btnJugador`, Text: "J" (jugador)
- Button → Name: `btnMeta`, Text: "M" (meta)

Código:

```
Public Class Form1

    Private Sub btnJugador_KeyDown(sender As Object, e As KeyEventArgs) Handles
Me.KeyDown

        Select Case e.KeyCode

            Case Keys.Up : btnJugador.Top -= 5

            Case Keys.Down : btnJugador.Top += 5

            Case Keys.Left : btnJugador.Left -= 5

            Case Keys.Right : btnJugador.Left += 5

        End Select

        If btnJugador.Bounds.IntersectsWith(btnMeta.Bounds) Then

            MessageBox.Show("¡Has ganado!")

        End If

    End Sub

End Class
```

(Asegurarse de KeyPreview=True en el Form.)

Programa 97. Juego de dados con imágenes
Controles:

- PictureBox → Name: `picDado1`, `picDado2`
- Button → Name: `btnTirar`, Text: "Tirar Dados"

Código:

```
Public Class Form1

    Dim rnd As New Random()


    Private Sub btnTirar_Click(sender As Object, e As EventArgs) Handles
btnTirar.Click

        Dim d1 As Integer = rnd.Next(1, 7)

        Dim d2 As Integer = rnd.Next(1, 7)

        picDado1.Image = My.Resources.ResourceManager.GetObject("dado" & d1)

        picDado2.Image = My.Resources.ResourceManager.GetObject("dado" & d2)

    End Sub

End Class
```

(Se requiere tener imágenes `dado1` a `dado6` en Resources.)

Programa 98. Juego de ahorcado
Controles:

- Label → Name: `lblPalabra`
- TextBox → Name: `txtLetra`
- Button → Name: `btnProbar`, Text: "Probar"
- Label → Name: `lblIntentos`

Código:

```
Public Class Form1
    Dim palabra As String = "VBNET"
    Dim adivinada As String = "_____"
    Dim intentos As Integer = 6

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        lblPalabra.Text = adivinada
        lblIntentos.Text = "Intentos: " & intentos
    End Sub

    Private Sub btnProbar_Click(sender As Object, e As EventArgs) Handles
btnProbar.Click
        Dim letra As String = txtLetra.Text.ToUpper()
        Dim nueva As Char() = adivinada.ToCharArray()
        For i As Integer = 0 To palabra.Length - 1
            If palabra(i) = letra Then
                nueva(i) = letra
            End If
        Next
        adivinada = New String(nueva)
        lblPalabra.Text = adivinada
        If Not palabra.Contains(letra) Then
            intentos -= 1
            lblIntentos.Text = "Intentos: " & intentos
        End If
        If adivinada = palabra Then MessageBox.Show("¡Ganaste!")
        If intentos = 0 Then MessageBox.Show("¡Perdiste!")
        txtLetra.Clear()
    End Sub
```

```
End Class
```

Programa 99. Juego de tic-tac-toe
Controles:

- 9 Buttons → Name: `btn1` a `btn9`, Text: ""

Código:

```
Public Class Form1

    Dim turno As Boolean = True ' True = X, False = O


    Private Sub btn_Click(sender As Object, e As EventArgs) Handles btn1.Click, btn2.Click, btn3.Click, btn4.Click, btn5.Click, btn6.Click, btn7.Click, btn8.Click, btn9.Click

        Dim btn As Button = CType(sender, Button)

        If btn.Text = "" Then

            btn.Text = If(turno, "X", "O")

            turno = Not turno

            ' Opcional: agregar función para revisar ganador

        End If

    End Sub

End Class
```

Programa 100. Juego de cartas simple
Controles:

- PictureBox → Name: `picCarta`
- Button → Name: `btnSacar`, Text: "Sacar Carta"

Código:

```
Public Class Form1

    Dim rnd As New Random()

    Dim cartas() As String = {"c1", "c2", "c3", "c4", "c5"}


    Private Sub btnSacar_Click(sender As Object, e As EventArgs) Handles btnSacar.Click

        Dim index As Integer = rnd.Next(cartas.Length)

        picCarta.Image = My.Resources.ResourceManager.GetObject(cartas(index))

    End Sub

End Class
```

(Se requiere agregar imágenes `c1` a `c5` en Resources.)

Programa 101. Juego de lotería con números aleatorios
Controles:

- Button → Name: `btnGenerar`, Text: "Generar Números"
- ListBox → Name: `lstNumeros`

Código:

```
Public Class Form1

    Dim rnd As New Random()


    Private Sub btnGenerar_Click(sender As Object, e As EventArgs) Handles
btnGenerar.Click

        lstNumeros.Items.Clear()

        For i As Integer = 1 To 6

            lstNumeros.Items.Add(rnd.Next(1, 50))

        Next

    End Sub

End Class
```

Programa 102. Juego de ruleta simple
Controles:

- Button → Name: `btnGirar`, Text: "Girar"
- Label → Name: `lblResultado`

Código:

```
Public Class Form1

    Dim rnd As New Random()


    Private Sub btnGirar_Click(sender As Object, e As EventArgs) Handles
btnGirar.Click

        Dim numero As Integer = rnd.Next(0, 37)

        lblResultado.Text = "Número: " & numero

    End Sub

End Class
```

Programa 103. Juego de preguntas y respuestas
Controles:

- Label → Name: `lblPregunta`
- TextBox → Name: `txtRespuesta`
- Button → Name: `btnVerificar`, Text: "Verificar"
- Label → Name: `lblResultado`

Código:

```
Public Class Form1
    Dim pregunta As String = "Capital de Francia?"
    Dim respuestaCorrecta As String = "París"

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        lblPregunta.Text = pregunta
    End Sub

    Private Sub btnVerificar_Click(sender As Object, e As EventArgs) Handles
btnVerificar.Click
        If txtRespuesta.Text.ToLower() = respuestaCorrecta.ToLower() Then
            lblResultado.Text = "Correcto"
        Else
            lblResultado.Text = "Incorrecto"
        End If
    End Sub
End Class
```

Programa 104. Juego de carreras con imágenes
Controles:

- PictureBox → Name: `picCarro1`, `picCarro2`
- Button → Name: `btnCorrer`, Text: "Correr"
- Timer → Name: `Timer1`, Interval: 50

Código:

```
Public Class Form1
    Dim rnd As New Random()

    Private Sub btnCorrer_Click(sender As Object, e As EventArgs) Handles
btnCorrer.Click
        Timer1.Start()
    End Sub

    Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
        picCarro1.Left += rnd.Next(1, 10)
        picCarro2.Left += rnd.Next(1, 10)
        If picCarro1.Right >= Me.Width Or picCarro2.Right >= Me.Width Then
            Timer1.Stop()
            MessageBox.Show("Carrera terminada")
        End If
    End Sub
End Class
```

Programa 105. Juego de disparos con botones
Controles:

- Button → Name: `btnDisparar`, Text: "Disparar"
- Label → Name: `lblPuntaje`

Código:

```
Public Class Form1

    Dim puntaje As Integer = 0


    Private Sub btnDisparar_Click(sender As Object, e As EventArgs) Handles
btnDisparar.Click

        If MessageBox.Show("Acertaste?", "Disparo", MessageBoxButtons.YesNo) =
DialogResult.Yes Then

            puntaje += 1

        End If

        lblPuntaje.Text = "Puntaje: " & puntaje

    End Sub

End Class
```

Programa 106. Juego de laberinto con teclas de dirección
Controles:

- Button → Name: `btnJugador`, Text: "J"
- Panel → Name: `pnlLaberinto` (paredes)
- Button → Name: `btnMeta`, Text: "M"

Código:

```
Public Class Form1

    Private Sub Form1_KeyDown(sender As Object, e As KeyEventArgs) Handles
Me.KeyDown

        Select Case e.KeyCode

            Case Keys.Up : btnJugador.Top -= 5

            Case Keys.Down : btnJugador.Top += 5

            Case Keys.Left : btnJugador.Left -= 5

            Case Keys.Right : btnJugador.Left += 5

        End Select

        If btnJugador.Bounds.IntersectsWith(btnMeta.Bounds) Then

            MessageBox.Show("¡Ganaste!")

        End If

    End Sub

End Class
```

(KeyPreview=True en el Form.)

Programa 107. Agenda con búsqueda por nombre
Controles:

- TextBox → Name: `txtBuscar`
- Button → Name: `btnBuscar`, Text: "Buscar"
- ListBox → Name: `lstContactos`

Código:

```
Public Class Form1
    Dim contactos As New List(Of String) From {"Ana", "Luis", "Pedro"}

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        lstContactos.Items.AddRange(contactos.ToArray())
    End Sub

    Private Sub btnBuscar_Click(sender As Object, e As EventArgs) Handles
btnBuscar.Click
        Dim filtro = txtBuscar.Text.ToLower()
        lstContactos.Items.Clear()
        For Each c In contactos
            If c.ToLower().Contains(filtro) Then lstContactos.Items.Add(c)
        Next
    End Sub
End Class
```

Programa 108. Agenda con edición de contactos
Controles:

- TextBox → Name: `txtContacto`
- Button → Name: `btnEditar`, Text: "Editar"
- ListBox → Name: `lstContactos`

Código:

```
Public Class Form1

    Private Sub btnEditar_Click(sender As Object, e As EventArgs) Handles
btnEditar.Click

        If lstContactos.SelectedIndex >= 0 Then

            lstContactos.Items(lstContactos.SelectedIndex) = txtContacto.Text

        End If

    End Sub

End Class
```

Programa 109. Agenda con borrado de contactos
Controles:

- Button → Name: `btnBorrar`, Text: "Borrar"
- ListBox → Name: `lstContactos`

Código:

```
Public Class Form1

    Private Sub btnBorrar_Click(sender As Object, e As EventArgs) Handles
btnBorrar.Click

        If lstContactos.SelectedIndex >= 0 Then

            lstContactos.Items.RemoveAt(lstContactos.SelectedIndex)

        End If

    End Sub

End Class
```

Programa 110. Agenda con exportación a archivo de texto
Controles:

- Button → Name: `btnExportar`, Text: "Exportar"
- ListBox → Name: `lstContactos`

Código:

```
Imports System.IO


Public Class Form1

    Private Sub btnExportar_Click(sender As Object, e As EventArgs) Handles
btnExportar.Click

        Using sw As New StreamWriter("agenda.txt")

            For Each c As String In lstContactos.Items

                sw.WriteLine(c)

            Next

        End Using

        MessageBox.Show("Exportado correctamente")

    End Sub

End Class
```

Programa 111. Agenda con importación desde archivo de texto
Controles:

- Button → Name: `btnImportar`, Text: "Importar"
- ListBox → Name: `lstContactos`
- OpenFileDialog → Name: `OpenFileDialog1`

Código:

```
Imports System.IO


Public Class Form1

    Private Sub btnImportar_Click(sender As Object, e As EventArgs) Handles
btnImportar.Click

        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then

            lstContactos.Items.Clear()

            Dim lineas() As String = File.ReadAllLines(OpenFileDialog1.FileName)

            lstContactos.Items.AddRange(lineas)

        End If

    End Sub

End Class
```

Programa 112. Agenda con exportación a Excel
Controles:

- Button → Name: `btnExportarExcel`, Text: "Exportar a Excel"
- ListBox → Name: `lstContactos`

Código:

```
Imports Microsoft.Office.Interop.Excel


Public Class Form1

    Private Sub btnExportarExcel_Click(sender As Object, e As EventArgs) Handles
btnExportarExcel.Click

        Dim app As New Application

        Dim wb = app.Workbooks.Add()

        Dim ws = CType(wb.Sheets(1), Worksheet)

        For i As Integer = 0 To lstContactos.Items.Count - 1

            ws.Cells(i + 1, 1) = lstContactos.Items(i)

        Next

        app.Visible = True

    End Sub

End Class
```

(Requiere referencia a Microsoft Excel Object Library.)

Programa 113. Agenda con importación desde Excel
Controles:

- Button → Name: `btnImportarExcel`, Text: "Importar desde Excel"
- ListBox → Name: `lstContactos`
- OpenFileDialog → Name: `OpenFileDialog1`

Código:

```
Imports Microsoft.Office.Interop.Excel


Public Class Form1

    Private Sub btnImportarExcel_Click(sender As Object, e As EventArgs) Handles
btnImportarExcel.Click

        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then

            Dim app As New Application

            Dim wb = app.Workbooks.Open(OpenFileDialog1.FileName)

            Dim ws = CType(wb.Sheets(1), Worksheet)

            lstContactos.Items.Clear()

            Dim fila As Integer = 1

            While ws.Cells(fila, 1).Value IsNot Nothing

                lstContactos.Items.Add(ws.Cells(fila, 1).Value)

                fila += 1

            End While

            wb.Close(False)

            app.Quit()

        End If

    End Sub

End Class
```

Programa 114. Agenda con base de datos SQLite
Controles:

- Button → Name: `btnCargar`, Text: "Cargar"
- Button → Name: `btnAgregar`, Text: "Agregar"
- TextBox → Name: `txtContacto`
- ListBox → Name: `lstContactos`

Código:

```
Imports System.Data.SQLite


Public Class Form1
    Dim conn As New SQLiteConnection("Data Source=agenda.db;Version=3;")


    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        conn.Open()
        Dim cmd As New SQLiteCommand("CREATE TABLE IF NOT EXISTS contactos
(nombre TEXT)", conn)
        cmd.ExecuteNonQuery()
        CargarContactos()
    End Sub


    Private Sub CargarContactos()
        lstContactos.Items.Clear()
        Dim cmd As New SQLiteCommand("SELECT nombre FROM contactos", conn)
        Dim dr = cmd.ExecuteReader()
        While dr.Read()
            lstContactos.Items.Add(dr("nombre"))
        End While
    End Sub


    Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
        Dim cmd As New SQLiteCommand("INSERT INTO contactos(nombre)
VALUES(@nombre)", conn)
        cmd.Parameters.AddWithValue("@nombre", txtContacto.Text)
        cmd.ExecuteNonQuery()
        CargarContactos()
```

```
    End Sub
End Class
```

Programa 115. Agenda con base de datos SQL Server
Controles:

- Button → Name: `btnCargar`, Text: "Cargar"
- Button → Name: `btnAgregar`, Text: "Agregar"
- TextBox → Name: `txtContacto`
- ListBox → Name: `lstContactos`

Código:

```
Imports System.Data.SqlClient


Public Class Form1

    Dim conn As New
SqlConnection("Server=localhost;Database=Agenda;Trusted_Connection=True;")


    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load

        conn.Open()

        CargarContactos()

    End Sub


    Private Sub CargarContactos()

        lstContactos.Items.Clear()

        Dim cmd As New SqlCommand("SELECT nombre FROM contactos", conn)

        Dim dr = cmd.ExecuteReader()

        While dr.Read()

            lstContactos.Items.Add(dr("nombre"))

        End While

        dr.Close()

    End Sub


    Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click

        Dim cmd As New SqlCommand("INSERT INTO contactos(nombre)
VALUES(@nombre)", conn)

        cmd.Parameters.AddWithValue("@nombre", txtContacto.Text)

        cmd.ExecuteNonQuery()

        CargarContactos()

    End Sub
```

```
End Class
```

Programa 116. Agenda con base de datos MySQL
Controles:

- Button → Name: `btnCargar`, Text: "Cargar"
- Button → Name: `btnAgregar`, Text: "Agregar"
- TextBox → Name: `txtContacto`
- ListBox → Name: `lstContactos`

Código:

```
Imports MySql.Data.MySqlClient


Public Class Form1

    Dim conn As New
MySqlConnection("server=localhost;user=root;database=agenda;password=;")


    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load

        conn.Open()

        CargarContactos()

    End Sub


    Private Sub CargarContactos()

        lstContactos.Items.Clear()

        Dim cmd As New MySqlCommand("SELECT nombre FROM contactos", conn)

        Dim dr = cmd.ExecuteReader()

        While dr.Read()

            lstContactos.Items.Add(dr("nombre"))

        End While

        dr.Close()

    End Sub


    Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click

        Dim cmd As New MySqlCommand("INSERT INTO contactos(nombre)
VALUES(@nombre)", conn)

        cmd.Parameters.AddWithValue("@nombre", txtContacto.Text)

        cmd.ExecuteNonQuery()

        CargarContactos()

    End Sub
```

```
End Class
```

Programa 117. Agenda con base de datos PostgreSQL
Controles:

- Button → Name: `btnCargar`, Text: "Cargar"
- Button → Name: `btnAgregar`, Text: "Agregar"
- TextBox → Name: `txtContacto`
- ListBox → Name: `lstContactos`

Código:

```
Imports Npgsql


Public Class Form1
    Dim conn As New
NpgsqlConnection("Host=localhost;Username=postgres;Password=;Database=agenda")


    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        conn.Open()
        CargarContactos()
    End Sub


    Private Sub CargarContactos()
        lstContactos.Items.Clear()
        Dim cmd As New NpgsqlCommand("SELECT nombre FROM contactos", conn)
        Dim dr = cmd.ExecuteReader()
        While dr.Read()
            lstContactos.Items.Add(dr("nombre"))
        End While
        dr.Close()
    End Sub


    Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
        Dim cmd As New NpgsqlCommand("INSERT INTO contactos(nombre)
VALUES(@nombre)", conn)
        cmd.Parameters.AddWithValue("@nombre", txtContacto.Text)
        cmd.ExecuteNonQuery()
        CargarContactos()
    End Sub
```

```
End Class
```

Programa 118. Aplicación de notas con recordatorio
Controles:

- TextBox → Name: `txtNota`, Multiline: True
- Button → Name: `btnGuardar`, Text: "Guardar Nota"
- DateTimePicker → Name: `dtpRecordatorio`
- Timer → Name: `Timer1`, Interval: 1000

Código:

```
Public Class Form1
    Dim notas As New List(Of Tuple(Of String, DateTime))

    Private Sub btnGuardar_Click(sender As Object, e As EventArgs) Handles
btnGuardar.Click
        notas.Add(Tuple.Create(txtNota.Text, dtpRecordatorio.Value))
    End Sub

    Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
        For Each nota In notas
            If DateTime.Now >= nota.Item2 Then
                MessageBox.Show("Recordatorio: " & nota.Item1)
            End If
        Next
    End Sub
End Class
```

Programa 119. Aplicación de calendario simple
Controles:

- MonthCalendar → Name: `MonthCalendar1`
- Label → Name: `lblFecha`

Código:

```
Public Class Form1
    Private Sub MonthCalendar1_DateChanged(sender As Object, e As
DateRangeEventArgs) Handles MonthCalendar1.DateChanged
        lblFecha.Text = "Fecha seleccionada: " & e.Start.ToShortDateString()
    End Sub
End Class
```

Programa 120. Agenda con calendario integrado

Controles:

- MonthCalendar → Name: `mcAgenda`
- ListBox → Name: `lstNotas`
- TextBox → Name: `txtNota`
- Button → Name: `btnAgregarNota`, Text: "Agregar Nota"

Código:

```
Public Class Form1

    Dim notas As New Dictionary(Of Date, List(Of String))


    Private Sub btnAgregarNota_Click(sender As Object, e As EventArgs) Handles
btnAgregarNota.Click

        Dim fecha As Date = mcAgenda.SelectionStart

        If Not notas.ContainsKey(fecha) Then notas(fecha) = New List(Of String)

        notas(fecha).Add(txtNota.Text)

        ActualizarLista(fecha)

    End Sub


    Private Sub mcAgenda_DateChanged(sender As Object, e As DateRangeEventArgs)
Handles mcAgenda.DateChanged

        ActualizarLista(e.Start)

    End Sub


    Private Sub ActualizarLista(fecha As Date)

        lstNotas.Items.Clear()

        If notas.ContainsKey(fecha) Then

            For Each n In notas(fecha)

                lstNotas.Items.Add(n)

            Next

        End If

    End Sub

End Class
```

Programa 121. Agenda con alarma
Controles:

- TextBox → Name: `txtNota`
- DateTimePicker → Name: `dtpAlarma`
- Button → Name: `btnAgregar`, Text: "Agregar Alarma"
- Timer → Name: `Timer1`, Interval: 1000
- ListBox → Name: `lstAlarmas`

Código:

```
Public Class Form1

    Dim alarmas As New List(Of Tuple(Of String, DateTime))


    Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click

        alarmas.Add(Tuple.Create(txtNota.Text, dtpAlarma.Value))

        lstAlarmas.Items.Add(txtNota.Text & " - " &
dtpAlarma.Value.ToShortTimeString())

    End Sub


    Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick

        For Each alarma In alarmas

            If DateTime.Now >= alarma.Item2 Then

                MessageBox.Show("Alarma: " & alarma.Item1)

                alarmas.Remove(alarma)

                Exit For

            End If

        Next

    End Sub

End Class
```

Programa 122. Agenda con notificaciones
Controles:

- ListBox → Name: `lstNotas`
- Button → Name: `btnAgregar`, Text: "Agregar Nota"
- TextBox → Name: `txtNota`
- Timer → Name: `Timer1`, Interval: 60000 (1 minuto)

Código:

```
Public Class Form1
    Dim notas As New List(Of String)

    Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
        notas.Add(txtNota.Text)
        lstNotas.Items.Add(txtNota.Text)
    End Sub

    Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
        If notas.Count > 0 Then
            MessageBox.Show("Tienes " & notas.Count & " nota(s) pendiente(s)")
        End If
    End Sub
End Class
```

Programa 123. Agenda con múltiples formularios
Controles:

- Button → Name: `btnAbrirForm2`, Text: "Abrir Formulario 2"

Código:

```
Public Class Form1

    Private Sub btnAbrirForm2_Click(sender As Object, e As EventArgs) Handles
btnAbrirForm2.Click

        Dim f2 As New Form2()

        f2.Show()

    End Sub

End Class
```

(Form2 debe crearse como otro Formulario en el proyecto.)

Programa 124. Agenda con login de usuario
Controles:

- TextBox → Name: `txtUsuario`
- TextBox → Name: `txtPassword`, PasswordChar: "*"
- Button → Name: `btnLogin`, Text: "Login"
- Label → Name: `lblMensaje`

Código:

```
Public Class Form1

    Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click

        If txtUsuario.Text = "admin" And txtPassword.Text = "1234" Then

            lblMensaje.Text = "Acceso concedido"

        Else

            lblMensaje.Text = "Usuario o contraseña incorrecta"

        End If

    End Sub

End Class
```

Programa 125. Agenda con niveles de usuario
Controles:

- ComboBox → Name: `cmbNivel`
- TextBox → Name: `txtUsuario`
- Button → Name: `btnLogin`, Text: "Login"
- Label → Name: `lblMensaje`

Código:

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        cmbNivel.Items.AddRange(New String() {"Usuario", "Administrador"})
        cmbNivel.SelectedIndex = 0
    End Sub

    Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
        If cmbNivel.SelectedItem = "Administrador" Then
            lblMensaje.Text = "Acceso de administrador"
        Else
            lblMensaje.Text = "Acceso de usuario normal"
        End If
    End Sub
End Class
```

Programa 126. Agenda con encriptación de datos
Controles:

- TextBox → Name: `txtDato`
- Button → Name: `btnEncriptar`, Text: "Encriptar"
- Label → Name: `lblResultado`

Código:

```
Imports System.Text


Public Class Form1

    Private Sub btnEncriptar_Click(sender As Object, e As EventArgs) Handles
btnEncriptar.Click

        Dim bytes() As Byte = Encoding.UTF8.GetBytes(txtDato.Text)

        Dim encriptado As String = Convert.ToBase64String(bytes)

        lblResultado.Text = encriptado

    End Sub

End Class
```

Programa 127. Agenda con desencriptación de datos
Controles:

- TextBox → Name: `txtEncriptado`
- Button → Name: `btnDesencriptar`, Text: "Desencriptar"
- Label → Name: `lblResultado`

Código:

```
Imports System.Text


Public Class Form1

    Private Sub btnDesencriptar_Click(sender As Object, e As EventArgs) Handles
btnDesencriptar.Click

        Dim bytes() As Byte = Convert.FromBase64String(txtEncriptado.Text)

        Dim original As String = Encoding.UTF8.GetString(bytes)

        lblResultado.Text = original

    End Sub

End Class
```

Programa 128. Calculadora científica simple
Controles:

- TextBox → Name: `txtPantalla`
- Buttons → Números 0-9 y operaciones (+, -, *, /)
- Button → Name: `btnIgual`, Text: "="

Código:

```
Public Class Form1

    Dim operacion As String = ""

    Dim valor1 As Double


    Private Sub btnNumero_Click(sender As Object, e As EventArgs) Handles
btn0.Click, btn1.Click, btn2.Click, btn3.Click, btn4.Click, btn5.Click,
btn6.Click, btn7.Click, btn8.Click, btn9.Click

        Dim btn As Button = CType(sender, Button)

        txtPantalla.Text &= btn.Text

    End Sub


    Private Sub btnOperacion_Click(sender As Object, e As EventArgs) Handles
btnSumar.Click, btnRestar.Click, btnMultiplicar.Click, btnDividir.Click

        valor1 = Val(txtPantalla.Text)

        operacion = CType(sender, Button).Text

        txtPantalla.Clear()

    End Sub


    Private Sub btnIgual_Click(sender As Object, e As EventArgs) Handles
btnIgual.Click

        Dim valor2 As Double = Val(txtPantalla.Text)

        Select Case operacion

            Case "+" : txtPantalla.Text = valor1 + valor2

            Case "-" : txtPantalla.Text = valor1 - valor2

            Case "*" : txtPantalla.Text = valor1 * valor2

            Case "/" : txtPantalla.Text = valor1 / valor2

        End Select

    End Sub

End Class
```

Programa 129. Calculadora con funciones trigonométricas
Controles:

- TextBox → Name: `txtPantalla`
- Button → Name: `btnSin`, Text: "Sin"
- Button → Name: `btnCos`, Text: "Cos"
- Button → Name: `btnTan`, Text: "Tan"

Código:

```
Public Class Form1

    Private Sub btnSin_Click(sender As Object, e As EventArgs) Handles
btnSin.Click

        txtPantalla.Text = Math.Sin(Val(txtPantalla.Text))

    End Sub


    Private Sub btnCos_Click(sender As Object, e As EventArgs) Handles
btnCos.Click

        txtPantalla.Text = Math.Cos(Val(txtPantalla.Text))

    End Sub


    Private Sub btnTan_Click(sender As Object, e As EventArgs) Handles
btnTan.Click

        txtPantalla.Text = Math.Tan(Val(txtPantalla.Text))

    End Sub

End Class
```

Programa 130. Calculadora con funciones logarítmicas
Controles:

- TextBox → Name: `txtPantalla`
- Button → Name: `btnLog`, Text: "Log"
- Button → Name: `btnLn`, Text: "Ln"

Código:

```
Public Class Form1

    Private Sub btnLog_Click(sender As Object, e As EventArgs) Handles
btnLog.Click

        txtPantalla.Text = Math.Log10(Val(txtPantalla.Text))

    End Sub


    Private Sub btnLn_Click(sender As Object, e As EventArgs) Handles
btnLn.Click

        txtPantalla.Text = Math.Log(Val(txtPantalla.Text))

    End Sub

End Class
```

Programa 131. Calculadora con potencias y raíces
Controles:

- TextBox → Name: `txtPantalla`
- Button → Name: `btnPotencia`, Text: "^"
- Button → Name: `btnRaiz`, Text: "√"
- Button → Name: `btnIgual`, Text: "="

Código:

```
Public Class Form1
    Dim valor1 As Double
    Dim operacion As String

    Private Sub btnPotencia_Click(sender As Object, e As EventArgs) Handles
btnPotencia.Click
        valor1 = Val(txtPantalla.Text)
        operacion = "^"
        txtPantalla.Clear()
    End Sub

    Private Sub btnRaiz_Click(sender As Object, e As EventArgs) Handles
btnRaiz.Click
        valor1 = Val(txtPantalla.Text)
        txtPantalla.Text = Math.Sqrt(valor1)
    End Sub

    Private Sub btnIgual_Click(sender As Object, e As EventArgs) Handles
btnIgual.Click
        Dim valor2 As Double = Val(txtPantalla.Text)
        If operacion = "^" Then
            txtPantalla.Text = Math.Pow(valor1, valor2)
        End If
    End Sub
End Class
```

Programa 132. Calculadora con historial de operaciones
Controles:

- TextBox → Name: `txtPantalla`
- ListBox → Name: `lstHistorial`
- Botones numéricos y de operaciones (+, -, *, /)
- Button → Name: `btnIgual`, Text: "="

Código:

```
Public Class Form1
    Dim valor1 As Double
    Dim operacion As String

    Private Sub btnOperacion_Click(sender As Object, e As EventArgs) Handles
btnSumar.Click, btnRestar.Click, btnMultiplicar.Click, btnDividir.Click
        valor1 = Val(txtPantalla.Text)
        operacion = CType(sender, Button).Text
        txtPantalla.Clear()
    End Sub

    Private Sub btnIgual_Click(sender As Object, e As EventArgs) Handles
btnIgual.Click
        Dim valor2 As Double = Val(txtPantalla.Text)
        Dim resultado As Double = 0
        Select Case operacion
            Case "+" : resultado = valor1 + valor2
            Case "-" : resultado = valor1 - valor2
            Case "*" : resultado = valor1 * valor2
            Case "/" : resultado = valor1 / valor2
        End Select
        txtPantalla.Text = resultado
        lstHistorial.Items.Add(valor1 & " " & operacion & " " & valor2 & " = " &
resultado)
    End Sub
End Class
```

Programa 133. Calculadora con memoria de resultados
Controles:

- TextBox → Name: `txtPantalla`
- Button → Name: `btnMS`, Text: "MS" (Memory Store)
- Button → Name: `btnMR`, Text: "MR" (Memory Recall)
- Button → Name: `btnMC`, Text: "MC" (Memory Clear)

Código:

```
Public Class Form1

    Dim memoria As Double = 0


    Private Sub btnMS_Click(sender As Object, e As EventArgs) Handles
btnMS.Click

        memoria = Val(txtPantalla.Text)

    End Sub


    Private Sub btnMR_Click(sender As Object, e As EventArgs) Handles
btnMR.Click

        txtPantalla.Text = memoria

    End Sub


    Private Sub btnMC_Click(sender As Object, e As EventArgs) Handles
btnMC.Click

        memoria = 0

    End Sub

End Class
```

Programa 134. Juego de preguntas con puntuación
Controles:

- Label → Name: `lblPregunta`
- TextBox → Name: `txtRespuesta`
- Button → Name: `btnVerificar`, Text: "Verificar"
- Label → Name: `lblPuntaje`

Código:

```
Public Class Form1
    Dim preguntas As New List(Of String) From {"Capital de Francia?", "5+5?"}
    Dim respuestas As New List(Of String) From {"París", "10"}
    Dim indice As Integer = 0
    Dim puntaje As Integer = 0

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        lblPregunta.Text = preguntas(indice)
        lblPuntaje.Text = "Puntaje: " & puntaje
    End Sub

    Private Sub btnVerificar_Click(sender As Object, e As EventArgs) Handles
btnVerificar.Click
        If txtRespuesta.Text.ToLower() = respuestas(indice).ToLower() Then
puntaje += 1
        indice += 1
        If indice < preguntas.Count Then
            lblPregunta.Text = preguntas(indice)
            txtRespuesta.Clear()
        Else
            MessageBox.Show("Juego terminado. Puntaje: " & puntaje)
        End If
        lblPuntaje.Text = "Puntaje: " & puntaje
    End Sub
End Class
```

Programa 135. Juego de preguntas con tiempo límite
Controles:

- Label → Name: `lblPregunta`
- TextBox → Name: `txtRespuesta`
- Button → Name: `btnVerificar`, Text: "Verificar"
- Label → Name: `lblTiempo`
- Timer → Name: `Timer1`, Interval: 1000

Código:

```
Public Class Form1
    Dim tiempo As Integer = 10
    Dim pregunta As String = "Capital de España?"
    Dim respuestaCorrecta As String = "Madrid"

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        lblPregunta.Text = pregunta
        lblTiempo.Text = "Tiempo: " & tiempo
        Timer1.Start()
    End Sub

    Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
        tiempo -= 1
        lblTiempo.Text = "Tiempo: " & tiempo
        If tiempo = 0 Then
            Timer1.Stop()
            MessageBox.Show("Se acabó el tiempo")
        End If
    End Sub
End Class
```

Programa 136. Juego de preguntas con niveles
Controles:

- Label → Name: `lblPregunta`
- TextBox → Name: `txtRespuesta`
- Button → Name: `btnVerificar`, Text: "Verificar"
- ComboBox → Name: `cmbNivel`, Items: "Fácil", "Medio", "Difícil"

Código:

```
Public Class Form1
    Dim preguntasFacil As New List(Of String) From {"2+2?"}
    Dim respuestasFacil As New List(Of String) From {"4"}

    Private Sub cmbNivel_SelectedIndexChanged(sender As Object, e As EventArgs)
Handles cmbNivel.SelectedIndexChanged
        If cmbNivel.SelectedItem = "Fácil" Then
            lblPregunta.Text = preguntasFacil(0)
        End If
    End Sub
End Class
```

Programa 137. Juego de preguntas con base de datos
Controles:

- Label → Name: `lblPregunta`
- TextBox → Name: `txtRespuesta`
- Button → Name: `btnVerificar`, Text: "Verificar"

Código (SQLite ejemplo):

```
Imports System.Data.SQLite


Public Class Form1
    Dim conn As New SQLiteConnection("Data Source=preguntas.db;Version=3;")
    Dim cmd As SQLiteCommand
    Dim dr As SQLiteDataReader
    Dim preguntaActual As String = ""
    Dim respuestaActual As String = ""


    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        conn.Open()
        cmd = New SQLiteCommand("CREATE TABLE IF NOT EXISTS preguntas(pregunta
TEXT, respuesta TEXT)", conn)
        cmd.ExecuteNonQuery()
        cmd = New SQLiteCommand("SELECT * FROM preguntas LIMIT 1", conn)
        dr = cmd.ExecuteReader()
        If dr.Read() Then
            preguntaActual = dr("pregunta")
            respuestaActual = dr("respuesta")
            lblPregunta.Text = preguntaActual
        End If
        dr.Close()
    End Sub


    Private Sub btnVerificar_Click(sender As Object, e As EventArgs) Handles
btnVerificar.Click
        If txtRespuesta.Text.ToLower() = respuestaActual.ToLower() Then
MessageBox.Show("Correcto") Else MessageBox.Show("Incorrecto")
    End Sub
End Class
```

Programa 138. Juego de preguntas con imágenes
Controles:

- PictureBox → Name: `picPregunta`
- TextBox → Name: `txtRespuesta`
- Button → Name: `btnVerificar`, Text: "Verificar"

Código:

```
Public Class Form1
    Dim preguntas As New List(Of String) From {"imagen1.jpg"}
    Dim respuestas As New List(Of String) From {"Gato"}
    Dim indice As Integer = 0

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        picPregunta.Image = Image.FromFile(preguntas(indice))
    End Sub

    Private Sub btnVerificar_Click(sender As Object, e As EventArgs) Handles
btnVerificar.Click
        If txtRespuesta.Text.ToLower() = respuestas(indice).ToLower() Then
            MessageBox.Show("Correcto")
        Else
            MessageBox.Show("Incorrecto")
        End If
    End Sub
End Class
```

Programa 139. Juego de preguntas con sonidos
Controles:

- Button → Name: `btnReproducir`, Text: "Reproducir Sonido"
- TextBox → Name: `txtRespuesta`
- Button → Name: `btnVerificar`, Text: "Verificar"

Código:

```
Imports System.Media


Public Class Form1
    Dim sonido As String = "sonido1.wav"
    Dim respuestaCorrecta As String = "Campana"


    Private Sub btnReproducir_Click(sender As Object, e As EventArgs) Handles
btnReproducir.Click
        Dim player As New SoundPlayer(sonido)
        player.Play()
    End Sub


    Private Sub btnVerificar_Click(sender As Object, e As EventArgs) Handles
btnVerificar.Click
        If txtRespuesta.Text.ToLower() = respuestaCorrecta.ToLower() Then
MessageBox.Show("Correcto") Else MessageBox.Show("Incorrecto")
    End Sub
End Class
```

Programa 140. Juego de preguntas multijugador
Controles:

- Label → Name: `lblPregunta`
- TextBox → Name: `txtRespuesta1`
- TextBox → Name: `txtRespuesta2`
- Button → Name: `btnVerificar`, Text: "Verificar"
- Label → Name: `lblPuntaje`

Código:

```
Public Class Form1
    Dim preguntas As New List(Of String) From {"Capital de Italia?"}
    Dim respuestas As New List(Of String) From {"Roma"}
    Dim puntaje1 As Integer = 0
    Dim puntaje2 As Integer = 0

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        lblPregunta.Text = preguntas(0)
        lblPuntaje.Text = "Jugador1: 0 - Jugador2: 0"
    End Sub

    Private Sub btnVerificar_Click(sender As Object, e As EventArgs) Handles
btnVerificar.Click
        If txtRespuesta1.Text.ToLower() = respuestas(0).ToLower() Then puntaje1
+= 1
        If txtRespuesta2.Text.ToLower() = respuestas(0).ToLower() Then puntaje2
+= 1
        lblPuntaje.Text = "Jugador1: " & puntaje1 & " - Jugador2: " & puntaje2
    End Sub
End Class
```

Programa 141. Juego de preguntas con ranking
Controles:

- Label → Name: `lblPregunta`
- TextBox → Name: `txtRespuesta`
- Button → Name: `btnVerificar`, Text: "Verificar"
- ListBox → Name: `lstRanking`

Código:

```
Public Class Form1
    Dim preguntas As New List(Of String) From {"Capital de Alemania?"}
    Dim respuestas As New List(Of String) From {"Berlín"}
    Dim puntajes As New List(Of Integer)
    Dim indice As Integer = 0
    Dim puntajeActual As Integer = 0

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        lblPregunta.Text = preguntas(indice)
    End Sub

    Private Sub btnVerificar_Click(sender As Object, e As EventArgs) Handles
btnVerificar.Click
        If txtRespuesta.Text.ToLower() = respuestas(indice).ToLower() Then
puntajeActual += 1
        puntajes.Add(puntajeActual)
        lstRanking.Items.Clear()
        For Each p In puntajes
            lstRanking.Items.Add("Puntaje: " & p)
        Next
        indice += 1
        If indice < preguntas.Count Then lblPregunta.Text = preguntas(indice)
    End Sub
End Class
```

Programa 142. Juego de preguntas con exportación de resultados
Controles:

- Button → Name: `btnExportar`, Text: "Exportar Resultados"
- ListBox → Name: `lstResultados`

Código:

```
Imports System.IO


Public Class Form1

    Private Sub btnExportar_Click(sender As Object, e As EventArgs) Handles
btnExportar.Click

        Dim archivo As String = "resultados.txt"

        File.WriteAllLines(archivo, lstResultados.Items.Cast(Of String)())

        MessageBox.Show("Resultados exportados a " & archivo)

    End Sub

End Class
```

Programa 143. Juego de preguntas con importación de resultados
Controles:

- Button → Name: `btnImportar`, Text: "Importar Resultados"
- ListBox → Name: `lstResultados`
- OpenFileDialog → Name: `OpenFileDialog1`

Código:

```
Imports System.IO


Public Class Form1

    Private Sub btnImportar_Click(sender As Object, e As EventArgs) Handles
btnImportar.Click

        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then

            lstResultados.Items.Clear()


lstResultados.Items.AddRange(File.ReadAllLines(OpenFileDialog1.FileName))

        End If

    End Sub

End Class
```

Programa 144. Juego de preguntas con estadísticas
Controles:

- Label → Name: `lblPregunta`
- TextBox → Name: `txtRespuesta`
- Button → Name: `btnVerificar`, Text: "Verificar"
- Label → Name: `lblEstadisticas`

Código:

```
Public Class Form1
    Dim preguntas As New List(Of String) From {"2+2?", "Capital de Italia?"}
    Dim respuestas As New List(Of String) From {"4", "Roma"}
    Dim correctas As Integer = 0
    Dim intentos As Integer = 0
    Dim indice As Integer = 0

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        lblPregunta.Text = preguntas(indice)
    End Sub

    Private Sub btnVerificar_Click(sender As Object, e As EventArgs) Handles
btnVerificar.Click
        intentos += 1
        If txtRespuesta.Text.ToLower() = respuestas(indice).ToLower() Then
correctas += 1
        indice += 1
        If indice < preguntas.Count Then
            lblPregunta.Text = preguntas(indice)
        Else
            lblPregunta.Text = "Juego terminado"
        End If
        lblEstadisticas.Text = "Correctas: " & correctas & " / Intentos: " &
intentos
    End Sub
End Class
```

Programa 145. Juego de preguntas con gráficos
Controles:

- Chart → Name: `chartResultados`
- Button → Name: `btnActualizar`, Text: "Actualizar Gráfico"

Código:

```
Imports System.Windows.Forms.DataVisualization.Charting


Public Class Form1
    Dim correctas As Integer = 3
    Dim incorrectas As Integer = 2


    Private Sub btnActualizar_Click(sender As Object, e As EventArgs) Handles
btnActualizar.Click
        chartResultados.Series.Clear()
        chartResultados.Series.Add("Resultados")
        chartResultados.Series("Resultados").Points.AddXY("Correctas",
correctas)
        chartResultados.Series("Resultados").Points.AddXY("Incorrectas",
incorrectas)
    End Sub
End Class
```

Programa 146. Juego de preguntas con personalización
Controles:

- TextBox → Name: `txtPregunta`
- Button → Name: `btnAgregarPregunta`, Text: "Agregar Pregunta"
- ListBox → Name: `lstPreguntas`

Código:

```
Public Class Form1
    Private Sub btnAgregarPregunta_Click(sender As Object, e As EventArgs)
Handles btnAgregarPregunta.Click
        lstPreguntas.Items.Add(txtPregunta.Text)
    End Sub
End Class
```

Programa 147. Juego de preguntas con guardado automático
Controles:

- TextBox → Name: `txtRespuesta`
- Button → Name: `btnGuardar`, Text: "Guardar"
- Timer → Name: `Timer1`, Interval: 5000

Código:

```
Imports System.IO


Public Class Form1

    Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick

        File.AppendAllText("respuestas.txt", txtRespuesta.Text &
Environment.NewLine)

    End Sub


    Private Sub btnGuardar_Click(sender As Object, e As EventArgs) Handles
btnGuardar.Click

        File.AppendAllText("respuestas.txt", txtRespuesta.Text &
Environment.NewLine)

    End Sub

End Class
```

Programa 148. Juego de preguntas con recuperación de partida
Controles:

- Button → Name: `btnGuardar`, Text: "Guardar Partida"
- Button → Name: `btnCargar`, Text: "Cargar Partida"
- Label → Name: `lblPregunta`

Código:

```
Imports System.IO


Public Class Form1
    Dim preguntas As New List(Of String) From {"Capital de Francia?", "2+2?"}
    Dim indice As Integer = 0


    Private Sub btnGuardar_Click(sender As Object, e As EventArgs) Handles
btnGuardar.Click
        File.WriteAllText("partida.txt", indice.ToString())
    End Sub


    Private Sub btnCargar_Click(sender As Object, e As EventArgs) Handles
btnCargar.Click
        If File.Exists("partida.txt") Then
            indice = CInt(File.ReadAllText("partida.txt"))
            lblPregunta.Text = preguntas(indice)
        End If
    End Sub
End Class
```

Programa 149. Juego de preguntas con dificultad progresiva
Controles:

- Label → Name: `lblPregunta`
- TextBox → Name: `txtRespuesta`
- Button → Name: `btnVerificar`, Text: "Verificar"

Código:

```
Public Class Form1
    Dim preguntasFacil As New List(Of String) From {"2+2?"}
    Dim preguntasMedio As New List(Of String) From {"Capital de España?"}
    Dim preguntasDificil As New List(Of String) From {"Raíz cuadrada de 256?"}
    Dim nivel As Integer = 1

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        lblPregunta.Text = preguntasFacil(0)
    End Sub

    Private Sub btnVerificar_Click(sender As Object, e As EventArgs) Handles
btnVerificar.Click
        Select Case nivel
            Case 1 : lblPregunta.Text = preguntasMedio(0) : nivel += 1
            Case 2 : lblPregunta.Text = preguntasDificil(0) : nivel += 1
            Case 3 : lblPregunta.Text = "Juego terminado"
        End Select
    End Sub
End Class
```

Programa 150. Juego de preguntas con logros desbloqueables
Controles:

- Label → Name: `lblPregunta`
- TextBox → Name: `txtRespuesta`
- Button → Name: `btnVerificar`, Text: "Verificar"
- ListBox → Name: `lstLogros`

Código:

```
Public Class Form1
    Dim preguntas As New List(Of String) From {"2+2?", "Capital de Francia?"}
    Dim respuestas As New List(Of String) From {"4", "París"}
    Dim puntaje As Integer = 0

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        lblPregunta.Text = preguntas(0)
    End Sub

    Private Sub btnVerificar_Click(sender As Object, e As EventArgs) Handles
btnVerificar.Click
        Dim indice As Integer = lstLogros.Items.Count
        If txtRespuesta.Text.ToLower() = respuestas(indice).ToLower() Then
puntaje += 1
        If puntaje = 1 Then lstLogros.Items.Add("Primer acierto!")
        If puntaje = 2 Then lstLogros.Items.Add("Dos aciertos seguidos!")
    End Sub
End Class
```

Categoría 3: Programas avanzados (151–300)

Programa 151. Editor de imágenes simple

Controles:

- PictureBox → Name: `picImagen`, SizeMode: `StretchImage`
- Button → Name: `btnCargar`, Text: "Cargar Imagen"
- Button → Name: `btnGrises`, Text: "Convertir a Gris"
- OpenFileDialog → Name: `OpenFileDialog1`

Código:

```
Public Class Form1

    Private Sub btnCargar_Click(sender As Object, e As EventArgs) Handles
btnCargar.Click
        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
            picImagen.Image = Image.FromFile(OpenFileDialog1.FileName)
        End If
    End Sub


    Private Sub btnGrises_Click(sender As Object, e As EventArgs) Handles
btnGrises.Click
        If picImagen.Image IsNot Nothing Then
            Dim bmp As New Bitmap(picImagen.Image)
            For x As Integer = 0 To bmp.Width - 1
                For y As Integer = 0 To bmp.Height - 1
                    Dim c As Color = bmp.GetPixel(x, y)
                    Dim gris As Integer = CInt((c.R + c.G + c.B) / 3)
                    bmp.SetPixel(x, y, Color.FromArgb(gris, gris, gris))
                Next
            Next
            picImagen.Image = bmp
        End If
    End Sub
End Class
```

Programa 152. Editor de audio simple
Controles:

- Button → Name: `btnCargar`, Text: "Cargar Audio"
- Button → Name: `btnReproducir`, Text: "Reproducir"
- OpenFileDialog → Name: `OpenFileDialog1`

Código:

```
Imports System.Media


Public Class Form1
    Dim player As SoundPlayer


    Private Sub btnCargar_Click(sender As Object, e As EventArgs) Handles
btnCargar.Click
        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
            player = New SoundPlayer(OpenFileDialog1.FileName)
        End If
    End Sub


    Private Sub btnReproducir_Click(sender As Object, e As EventArgs) Handles
btnReproducir.Click
        If player IsNot Nothing Then player.Play()
    End Sub
End Class
```

Programa 153. Reproductor de música básico
Controles:

- Button → Name: `btnCargar`, Text: "Cargar Música"
- Button → Name: `btnPlay`, Text: "Play"
- Button → Name: `btnStop`, Text: "Stop"
- OpenFileDialog → Name: `OpenFileDialog1`
- AxWindowsMediaPlayer → Name: `wmp`

Código:

```
Public Class Form1

    Private Sub btnCargar_Click(sender As Object, e As EventArgs) Handles
btnCargar.Click

        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then

            wmp.URL = OpenFileDialog1.FileName

        End If

    End Sub


    Private Sub btnPlay_Click(sender As Object, e As EventArgs) Handles
btnPlay.Click

        wmp.Ctlcontrols.play()

    End Sub


    Private Sub btnStop_Click(sender As Object, e As EventArgs) Handles
btnStop.Click

        wmp.Ctlcontrols.stop()

    End Sub

End Class
```

Programa 154. Reproductor de video básico
Controles:

- Button → Name: `btnCargar`, Text: "Cargar Video"
- AxWindowsMediaPlayer → Name: `wmp`
- OpenFileDialog → Name: `OpenFileDialog1`

Código:

```
Public Class Form1

    Private Sub btnCargar_Click(sender As Object, e As EventArgs) Handles
btnCargar.Click

        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then

            wmp.URL = OpenFileDialog1.FileName

            wmp.Ctlcontrols.play()

        End If

    End Sub

End Class
```

Programa 155. Reproductor multimedia integrado
Controles:

- Button → Name: `btnCargarAudio`, Text: "Cargar Audio"
- Button → Name: `btnCargarVideo`, Text: "Cargar Video"
- AxWindowsMediaPlayer → Name: `wmp`
- OpenFileDialog → Name: `OpenFileDialog1`

Código:

```
Public Class Form1

    Private Sub btnCargarAudio_Click(sender As Object, e As EventArgs) Handles
btnCargarAudio.Click

        OpenFileDialog1.Filter = "WAV|*.wav|MP3|*.mp3"

        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then

            wmp.URL = OpenFileDialog1.FileName

        End If

    End Sub


    Private Sub btnCargarVideo_Click(sender As Object, e As EventArgs) Handles
btnCargarVideo.Click

        OpenFileDialog1.Filter = "MP4|*.mp4|AVI|*.avi"

        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then

            wmp.URL = OpenFileDialog1.FileName

        End If

    End Sub

End Class
```

Programa 156. Aplicación que controla volumen del sistema
Controles:

- TrackBar → Name: `trkVolumen`, Minimum: 0, Maximum: 100
- Label → Name: `lblVolumen`

Código:

```
Imports System.Runtime.InteropServices


Public Class Form1

    <DllImport("user32.dll")>

    Public Shared Function SendMessageW(ByVal hWnd As IntPtr, ByVal Msg As
Integer, ByVal wParam As Integer, ByVal lParam As Integer) As Integer

    End Function


    Private Sub trkVolumen_Scroll(sender As Object, e As EventArgs) Handles
trkVolumen.Scroll

        lblVolumen.Text = "Volumen: " & trkVolumen.Value

        ' Nota: Cambiar volumen global requiere API avanzada o bibliotecas
externas

    End Sub

End Class
```

Programa 157. Aplicación que controla brillo de pantalla
Controles:

- TrackBar → Name: `trkBrillo`, Minimum: 0, Maximum: 100
- Label → Name: `lblBrillo`

Código:

```
Public Class Form1

    Private Sub trkBrillo_Scroll(sender As Object, e As EventArgs) Handles
trkBrillo.Scroll

        lblBrillo.Text = "Brillo: " & trkBrillo.Value

        ' Nota: Ajustar brillo real requiere WMI o APIs específicas

    End Sub

End Class
```

Programa 158. Aplicación que toma capturas de pantalla
Controles:

- Button → Name: `btnCaptura`, Text: "Capturar Pantalla"
- PictureBox → Name: `picPantalla`, SizeMode: `StretchImage`

Código:

```
Public Class Form1

    Private Sub btnCaptura_Click(sender As Object, e As EventArgs) Handles
btnCaptura.Click

        Dim bmp As New Bitmap(Screen.PrimaryScreen.Bounds.Width,
Screen.PrimaryScreen.Bounds.Height)

        Dim g As Graphics = Graphics.FromImage(bmp)

        g.CopyFromScreen(0, 0, 0, 0, bmp.Size)

        picPantalla.Image = bmp

    End Sub

End Class
```

Programa 159. Aplicación que graba sonido
Controles:

- Button → Name: `btnGrabar`, Text: "Grabar"
- Button → Name: `btnDetener`, Text: "Detener"
- SaveFileDialog → Name: `SaveFileDialog1`

Código (usando NAudio):

```
Imports NAudio.Wave


Public Class Form1

    Dim waveIn As WaveInEvent

    Dim writer As WaveFileWriter


    Private Sub btnGrabar_Click(sender As Object, e As EventArgs) Handles
btnGrabar.Click

        waveIn = New WaveInEvent()

        waveIn.WaveFormat = New WaveFormat(44100, 1)

        waveIn.DataAvailable += Sub(s, a)

                                    If writer IsNot Nothing Then
writer.Write(a.Buffer, 0, a.BytesRecorded)

                                End Sub

        SaveFileDialog1.Filter = "WAV|*.wav"

        If SaveFileDialog1.ShowDialog() = DialogResult.OK Then

            writer = New WaveFileWriter(SaveFileDialog1.FileName,
waveIn.WaveFormat)

            waveIn.StartRecording()

        End If

    End Sub


    Private Sub btnDetener_Click(sender As Object, e As EventArgs) Handles
btnDetener.Click

        waveIn.StopRecording()

        writer.Close()

    End Sub

End Class
```

Programa 160. Aplicación que graba video
Controles:

- Button → Name: `btnGrabar`, Text: "Grabar Video"
- Button → Name: `btnDetener`, Text: "Detener"
- PictureBox → Name: `picVideo`

Código (ejemplo conceptual, requiere AForge o EmguCV):

```
Imports AForge.Video
Imports AForge.Video.DirectShow

Public Class Form1
    Dim videoSource As VideoCaptureDevice

    Private Sub btnGrabar_Click(sender As Object, e As EventArgs) Handles
btnGrabar.Click
        Dim videoDevices As New
FilterInfoCollection(FilterCategory.VideoInputDevice)
        videoSource = New VideoCaptureDevice(videoDevices(0).MonikerString)
        AddHandler videoSource.NewFrame, Sub(s, a) picVideo.Image =
DirectCast(a.Clone(), Bitmap)
        videoSource.Start()
    End Sub

    Private Sub btnDetener_Click(sender As Object, e As EventArgs) Handles
btnDetener.Click
        If videoSource IsNot Nothing AndAlso videoSource.IsRunning Then
            videoSource.SignalToStop()
        End If
    End Sub
End Class
```

Programa 161. Aplicación que reproduce listas de reproducción
Controles:

- ListBox → Name: `lstPlaylist`
- Button → Name: `btnAgregar`, Text: "Agregar Canción"
- Button → Name: `btnReproducir`, Text: "Reproducir"
- OpenFileDialog → Name: `OpenFileDialog1`
- AxWindowsMediaPlayer → Name: `wmp`

Código:

```
Public Class Form1

    Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click

        OpenFileDialog1.Filter = "MP3|*.mp3|WAV|*.wav"

        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then

            lstPlaylist.Items.Add(OpenFileDialog1.FileName)

        End If

    End Sub


    Private Sub btnReproducir_Click(sender As Object, e As EventArgs) Handles
btnReproducir.Click

        If lstPlaylist.SelectedItem IsNot Nothing Then

            wmp.URL = lstPlaylist.SelectedItem.ToString()

            wmp.Ctlcontrols.play()

        End If

    End Sub

End Class
```

Programa 162. Aplicación que descarga archivos de internet
Controles:

- TextBox → Name: `txtURL`
- Button → Name: `btnDescargar`, Text: "Descargar"
- SaveFileDialog → Name: `SaveFileDialog1`

Código:

```
Imports System.Net


Public Class Form1

    Private Sub btnDescargar_Click(sender As Object, e As EventArgs) Handles
btnDescargar.Click

        If SaveFileDialog1.ShowDialog() = DialogResult.OK Then

            Dim client As New WebClient()

            client.DownloadFile(txtURL.Text, SaveFileDialog1.FileName)

            MessageBox.Show("Descarga completada")

        End If

    End Sub

End Class
```

Programa 163. Aplicación que sube archivos a servidor FTP
Controles:

- TextBox → Name: `txtServidor`, Text: "ftp://servidor.com/"
- TextBox → Name: `txtUsuario`
- TextBox → Name: `txtPassword`
- Button → Name: `btnSubir`, Text: "Subir Archivo"
- OpenFileDialog → Name: `OpenFileDialog1`

Código:

```
Imports System.Net


Public Class Form1

    Private Sub btnSubir_Click(sender As Object, e As EventArgs) Handles
btnSubir.Click

        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then

            Dim request As FtpWebRequest =
CType(WebRequest.Create(txtServidor.Text &
IO.Path.GetFileName(OpenFileDialog1.FileName)), FtpWebRequest)

            request.Method = WebRequestMethods.Ftp.UploadFile

            request.Credentials = New NetworkCredential(txtUsuario.Text,
txtPassword.Text)

            Dim fileContents() As Byte =
IO.File.ReadAllBytes(OpenFileDialog1.FileName)

            request.ContentLength = fileContents.Length

            Dim requestStream = request.GetRequestStream()

            requestStream.Write(fileContents, 0, fileContents.Length)

            requestStream.Close()

            MessageBox.Show("Archivo subido correctamente")

        End If

    End Sub

End Class
```

Programa 164. Aplicación que se conecta a una API REST
Controles:

- TextBox → Name: `txtURL`
- Button → Name: `btnConsultar`, Text: "Consultar"
- TextBox → Name: `txtResultado`, Multiline: True, ScrollBars: `Vertical`

Código:

```
Imports System.Net


Public Class Form1

    Private Sub btnConsultar_Click(sender As Object, e As EventArgs) Handles
btnConsultar.Click

        Dim client As New WebClient()

        Dim response As String = client.DownloadString(txtURL.Text)

        txtResultado.Text = response

    End Sub

End Class
```

Programa 165. Aplicación que se conecta a una API SOAP
Controles:

- TextBox → Name: `txtServicio`
- Button → Name: `btnConsultar`, Text: "Consultar"
- TextBox → Name: `txtResultado`, Multiline: True, ScrollBars: `Vertical`

Código (ejemplo conceptual, requiere referencia SOAP agregada):

```
Public Class Form1

    Private Sub btnConsultar_Click(sender As Object, e As EventArgs) Handles
btnConsultar.Click

        Dim client As New ServiceReference1.MiServicioSoapClient()

        txtResultado.Text = client.MiMetodo("parametro")

    End Sub

End Class
```

Programa 166. Aplicación que consume servicios web JSON
Controles:

- TextBox → Name: `txtURL`
- Button → Name: `btnConsultar`, Text: "Consultar"
- TextBox → Name: `txtResultado`, Multiline: True

Código:

```
Imports System.Net

Imports Newtonsoft.Json.Linq


Public Class Form1

    Private Sub btnConsultar_Click(sender As Object, e As EventArgs) Handles
btnConsultar.Click

        Dim client As New WebClient()

        Dim json As String = client.DownloadString(txtURL.Text)

        Dim obj = JObject.Parse(json)

        txtResultado.Text = obj.ToString()

    End Sub

End Class
```

Programa 167. Aplicación que consume servicios web XML
Controles:

- TextBox → Name: `txtURL`
- Button → Name: `btnConsultar`, Text: "Consultar"
- TextBox → Name: `txtResultado`, Multiline: True

Código:

```
Imports System.Net
Imports System.Xml


Public Class Form1
    Private Sub btnConsultar_Click(sender As Object, e As EventArgs) Handles btnConsultar.Click
        Dim client As New WebClient()
        Dim xmlString As String = client.DownloadString(txtURL.Text)
        Dim doc As New XmlDocument()
        doc.LoadXml(xmlString)
        txtResultado.Text = doc.OuterXml
    End Sub
End Class
```

Programa 168. Cliente de chat local simple
Controles:

- TextBox → Name: `txtMensaje`
- Button → Name: `btnEnviar`, Text: "Enviar"
- ListBox → Name: `lstChat`

Código:

```
Public Class Form1
    Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles btnEnviar.Click
        lstChat.Items.Add("Tú: " & txtMensaje.Text)
        txtMensaje.Clear()
    End Sub
End Class
```

Programa 169. Cliente de chat en red local
Controles:

- TextBox → Name: `txtMensaje`
- Button → Name: `btnEnviar`, Text: "Enviar"
- ListBox → Name: `lstChat`

Código (ejemplo conceptual usando TcpClient):

```
Imports System.Net.Sockets
Imports System.Text
Imports System.Threading

Public Class Form1
    Dim client As TcpClient
    Dim stream As NetworkStream

    Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles
btnEnviar.Click
        If client IsNot Nothing Then
            Dim msg() As Byte = Encoding.ASCII.GetBytes(txtMensaje.Text)
            stream.Write(msg, 0, msg.Length)
            lstChat.Items.Add("Tú: " & txtMensaje.Text)
            txtMensaje.Clear()
        End If
    End Sub
End Class
```

Programa 170. Cliente de chat con base de datos
Controles:

- TextBox → Name: `txtMensaje`
- Button → Name: `btnEnviar`, Text: "Enviar"
- ListBox → Name: `lstChat`

Código (ejemplo conceptual usando SQLite):

```
Imports System.Data.SQLite


Public Class Form1

    Dim conn As New SQLiteConnection("Data Source=chat.db;Version=3;")

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load

        conn.Open()

        Dim cmd As New SQLiteCommand("CREATE TABLE IF NOT EXISTS
mensajes(usuario TEXT, mensaje TEXT)", conn)

        cmd.ExecuteNonQuery()

    End Sub


    Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles
btnEnviar.Click

        Dim cmd As New SQLiteCommand("INSERT INTO mensajes(usuario,mensaje)
VALUES('Tú',@msg)", conn)

        cmd.Parameters.AddWithValue("@msg", txtMensaje.Text)

        cmd.ExecuteNonQuery()

        lstChat.Items.Add("Tú: " & txtMensaje.Text)

        txtMensaje.Clear()

    End Sub

End Class
```

Programa 171. Cliente de chat con login de usuario
Controles:

- TextBox → Name: `txtUsuario`
- TextBox → Name: `txtContraseña`, PasswordChar: *
- Button → Name: `btnLogin`, Text: "Iniciar Sesión"
- ListBox → Name: `lstChat`

Código:

```
Public Class Form1
    Dim usuarioLogueado As String = ""

    Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
        If txtUsuario.Text <> "" And txtContraseña.Text <> "" Then
            usuarioLogueado = txtUsuario.Text
            MessageBox.Show("Bienvenido " & usuarioLogueado)
        Else
            MessageBox.Show("Ingrese usuario y contraseña")
        End If
    End Sub
End Class
```

Programa 172. Cliente de chat con historial de mensajes
Controles:

- TextBox → Name: `txtMensaje`
- Button → Name: `btnEnviar`, Text: "Enviar"
- ListBox → Name: `lstChat`

Código:

```
Imports System.IO


Public Class Form1
    Dim historialArchivo As String = "historial.txt"


    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        If File.Exists(historialArchivo) Then
            lstChat.Items.AddRange(File.ReadAllLines(historialArchivo))
        End If
    End Sub


    Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles
btnEnviar.Click
        lstChat.Items.Add("Tú: " & txtMensaje.Text)
        File.AppendAllText(historialArchivo, "Tú: " & txtMensaje.Text &
Environment.NewLine)
        txtMensaje.Clear()
    End Sub
End Class
```

Programa 173. Cliente de chat con emoticonos
Controles:

- TextBox → Name: `txtMensaje`
- Button → Name: `btnEnviar`, Text: "Enviar"
- ComboBox → Name: `cmbEmoticonos` (Items: 😀, 😳, 😎)
- ListBox → Name: `lstChat`

Código:

```
Public Class Form1

    Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles
btnEnviar.Click

        Dim emoticono As String = cmbEmoticonos.SelectedItem

        lstChat.Items.Add("Tú: " & txtMensaje.Text & " " & emoticono)

        txtMensaje.Clear()

    End Sub

End Class
```

Programa 174. Cliente de chat con archivos adjuntos
Controles:

- TextBox → Name: `txtMensaje`
- Button → Name: `btnEnviar`, Text: "Enviar"
- Button → Name: `btnAdjuntar`, Text: "Adjuntar Archivo"
- OpenFileDialog → Name: `OpenFileDialog1`
- ListBox → Name: `lstChat`

Código:

```
Public Class Form1
    Dim archivoAdjunto As String = ""

    Private Sub btnAdjuntar_Click(sender As Object, e As EventArgs) Handles
btnAdjuntar.Click
        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
            archivoAdjunto = OpenFileDialog1.FileName
            MessageBox.Show("Archivo adjuntado: " & archivoAdjunto)
        End If
    End Sub

    Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles
btnEnviar.Click
        lstChat.Items.Add("Tú: " & txtMensaje.Text & " " & archivoAdjunto)
        txtMensaje.Clear()
        archivoAdjunto = ""
    End Sub
End Class
```

Programa 175. Cliente de chat con múltiples salas
Controles:

- ComboBox → Name: `cmbSalas` (Items: General, Juegos, Música)
- TextBox → Name: `txtMensaje`
- Button → Name: `btnEnviar`, Text: "Enviar"
- ListBox → Name: `lstChat`

Código:

```
Dim salas As New Dictionary(Of String, List(Of String)) From {
    {"General", New List(Of String)}, {"Juegos", New List(Of String)},
{"Música", New List(Of String)}
}


Private Sub cmbSalas_SelectedIndexChanged(sender As Object, e As EventArgs)
Handles cmbSalas.SelectedIndexChanged
    lstChat.Items.Clear()
    lstChat.Items.AddRange(salas(cmbSalas.SelectedItem).ToArray())
End Sub


Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles
btnEnviar.Click
    Dim sala As String = cmbSalas.SelectedItem
    salas(sala).Add("Tú: " & txtMensaje.Text)
    lstChat.Items.Add("Tú: " & txtMensaje.Text)
    txtMensaje.Clear()
End Sub
```

Programa 176. Cliente de chat con videollamadas
Controles:

- Button → Name: `btnIniciarVideo`, Text: "Iniciar Videollamada"
- PictureBox → Name: `picVideo`
- AxWindowsMediaPlayer → Name: `wmpVideo`

Código conceptual:

```
Public Class Form1

    Private Sub btnIniciarVideo_Click(sender As Object, e As EventArgs) Handles
btnIniciarVideo.Click

        MessageBox.Show("Videollamada iniciada (implementación con AForge o
WebRTC necesaria)")

    End Sub

End Class
```

Programa 177. Cliente de chat con llamadas de voz
Controles:

- Button → Name: `btnIniciarVoz`, Text: "Iniciar Llamada"
- Button → Name: `btnDetenerVoz`, Text: "Detener"

Código conceptual:

```
Public Class Form1

    Private Sub btnIniciarVoz_Click(sender As Object, e As EventArgs) Handles
btnIniciarVoz.Click

        MessageBox.Show("Llamada de voz iniciada (implementación con NAudio o
VoIP necesaria)")

    End Sub


    Private Sub btnDetenerVoz_Click(sender As Object, e As EventArgs) Handles
btnDetenerVoz.Click

        MessageBox.Show("Llamada de voz detenida")

    End Sub

End Class
```

Programa 178. Cliente de chat con notificaciones
Controles:

- NotifyIcon → Name: `NotifyIcon1`, Icon: `icono.ico`
- Button → Name: `btnNotificar`, Text: "Enviar Notificación"

Código:

```
Public Class Form1

    Private Sub btnNotificar_Click(sender As Object, e As EventArgs) Handles
btnNotificar.Click

        NotifyIcon1.Visible = True

        NotifyIcon1.ShowBalloonTip(3000, "Chat", "Tienes un nuevo mensaje",
ToolTipIcon.Info)

    End Sub

End Class
```

Programa 179. Cliente de chat con moderadores
Controles:

- ListBox → Name: `lstChat`
- Button → Name: `btnBloquear`, Text: "Bloquear Usuario"
- TextBox → Name: `txtUsuario`

Código conceptual:

```
Dim usuariosBloqueados As New List(Of String)


Private Sub btnBloquear_Click(sender As Object, e As EventArgs) Handles
btnBloquear.Click

    usuariosBloqueados.Add(txtUsuario.Text)

    MessageBox.Show("Usuario " & txtUsuario.Text & " bloqueado")

End Sub
```

Programa 180. Cliente de chat con administración
Controles:

- ListBox → Name: `lstUsuarios`
- Button → Name: `btnEliminar`, Text: "Eliminar Usuario"
- Button → Name: `btnAgregar`, Text: "Agregar Usuario"
- TextBox → Name: `txtUsuario`

Código:

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click

    lstUsuarios.Items.Add(txtUsuario.Text)

End Sub


Private Sub btnEliminar_Click(sender As Object, e As EventArgs) Handles
btnEliminar.Click

    If lstUsuarios.SelectedItem IsNot Nothing Then
lstUsuarios.Items.Remove(lstUsuarios.SelectedItem)

End Sub
```

Programa 181. Cliente de chat con cifrado de mensajes
Controles:

- TextBox → Name: `txtMensaje`
- Button → Name: `btnEnviar`, Text: "Enviar"
- ListBox → Name: `lstChat`

Código (ejemplo simple con Base64):

```
Imports System.Text


Public Class Form1

    Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles
btnEnviar.Click

        Dim mensaje As String = txtMensaje.Text

        Dim mensajeCifrado As String =
Convert.ToBase64String(Encoding.UTF8.GetBytes(mensaje))

        lstChat.Items.Add("Tú: " & mensajeCifrado)

        txtMensaje.Clear()

    End Sub

End Class
```

Programa 182. Cliente de chat con recuperación de historial
Controles:

- TextBox → Name: `txtMensaje`
- Button → Name: `btnEnviar`, Text: "Enviar"
- Button → Name: `btnRecuperar`, Text: "Recuperar Historial"
- ListBox → Name: `lstChat`

Código:

```
Imports System.IO


Public Class Form1
    Dim archivoHistorial As String = "historial_chat.txt"


    Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles
btnEnviar.Click
        lstChat.Items.Add("Tú: " & txtMensaje.Text)
        File.AppendAllText(archivoHistorial, "Tú: " & txtMensaje.Text &
Environment.NewLine)
        txtMensaje.Clear()
    End Sub


    Private Sub btnRecuperar_Click(sender As Object, e As EventArgs) Handles
btnRecuperar.Click
        If File.Exists(archivoHistorial) Then
            lstChat.Items.Clear()
            lstChat.Items.AddRange(File.ReadAllLines(archivoHistorial))
        End If
    End Sub
End Class
```

Programa 183. Cliente de chat con estadísticas
Controles:

- Button → Name: `btnEstadisticas`, Text: "Mostrar Estadísticas"
- ListBox → Name: `lstChat`

Código:

```
Public Class Form1

    Private Sub btnEstadisticas_Click(sender As Object, e As EventArgs) Handles
btnEstadisticas.Click

        Dim totalMensajes As Integer = lstChat.Items.Count

        MessageBox.Show("Total de mensajes: " & totalMensajes)

    End Sub

End Class
```

Programa 184. Cliente de chat con exportación de mensajes
Controles:

- Button → Name: `btnExportar`, Text: "Exportar"
- SaveFileDialog → Name: `SaveFileDialog1`
- ListBox → Name: `lstChat`

Código:

```
Imports System.IO


Public Class Form1

    Private Sub btnExportar_Click(sender As Object, e As EventArgs) Handles
btnExportar.Click

        If SaveFileDialog1.ShowDialog() = DialogResult.OK Then

            File.WriteAllLines(SaveFileDialog1.FileName, lstChat.Items.Cast(Of
String)())

            MessageBox.Show("Mensajes exportados")

        End If

    End Sub

End Class
```

Programa 185. Cliente de chat con importación de mensajes
Controles:

- Button → Name: `btnImportar`, Text: "Importar"
- OpenFileDialog → Name: `OpenFileDialog1`
- ListBox → Name: `lstChat`

Código:

```
Imports System.IO


Public Class Form1

    Private Sub btnImportar_Click(sender As Object, e As EventArgs) Handles
btnImportar.Click

        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then

            lstChat.Items.Clear()

            lstChat.Items.AddRange(File.ReadAllLines(OpenFileDialog1.FileName))

            MessageBox.Show("Mensajes importados")

        End If

    End Sub

End Class
```

Programa 186. Cliente de correo electrónico básico
Controles:

- TextBox → Name: `txtPara`
- TextBox → Name: `txtAsunto`
- TextBox → Name: `txtCuerpo`, Multiline: True
- Button → Name: `btnEnviar`, Text: "Enviar"

Código conceptual:

```
Imports System.Net.Mail


Public Class Form1

    Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles
btnEnviar.Click

        Dim mail As New MailMessage()

        mail.To.Add(txtPara.Text)

        mail.Subject = txtAsunto.Text

        mail.Body = txtCuerpo.Text

        MessageBox.Show("Correo preparado (implementación real requiere SMTP)")

    End Sub

End Class
```

Programa 187. Cliente de correo con envío SMTP
Controles:

- TextBox → Name: `txtPara`
- TextBox → Name: `txtAsunto`
- TextBox → Name: `txtCuerpo`, Multiline: True
- TextBox → Name: `txtSMTP`
- TextBox → Name: `txtUsuario`
- TextBox → Name: `txtPassword`, PasswordChar: *
- Button → Name: `btnEnviar`, Text: "Enviar"

Código:

```
Imports System.Net.Mail


Public Class Form1

    Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles
btnEnviar.Click

        Dim smtp As New SmtpClient(txtSMTP.Text)

        smtp.Credentials = New Net.NetworkCredential(txtUsuario.Text,
txtPassword.Text)

        Dim mail As New MailMessage(txtUsuario.Text, txtPara.Text,
txtAsunto.Text, txtCuerpo.Text)

        smtp.Send(mail)

        MessageBox.Show("Correo enviado")

    End Sub

End Class
```

Programa 188. Cliente de correo con recepción POP3
Controles:

- TextBox → Name: `txtServidor`
- TextBox → Name: `txtUsuario`
- TextBox → Name: `txtPassword`, PasswordChar: *
- Button → Name: `btnRecibir`, Text: "Recibir"
- ListBox → Name: `lstCorreos`

Código conceptual usando OpenPop.NET:

```
Imports OpenPop.Pop3


Public Class Form1

    Private Sub btnRecibir_Click(sender As Object, e As EventArgs) Handles
btnRecibir.Click

        Dim client As New Pop3Client()

        client.Connect(txtServidor.Text, 110, False)

        client.Authenticate(txtUsuario.Text, txtPassword.Text)

        Dim count = client.GetMessageCount()

        lstCorreos.Items.Clear()

        For i As Integer = 1 To count

            Dim msg = client.GetMessage(i)

            lstCorreos.Items.Add(msg.Headers.Subject)

        Next

    End Sub

End Class
```

Programa 189. Cliente de correo con conexión IMAP
Controles:

- TextBox → Name: `txtServidor`
- TextBox → Name: `txtUsuario`
- TextBox → Name: `txtPassword`, PasswordChar: *
- Button → Name: `btnRecibir`, Text: "Recibir"
- ListBox → Name: `lstCorreos`

Código conceptual usando MailKit:

```
Imports MailKit.Net.Imap

Imports MailKit

Imports MimeKit


Public Class Form1

    Private Sub btnRecibir_Click(sender As Object, e As EventArgs) Handles
btnRecibir.Click

        Using client As New ImapClient()

            client.Connect(txtServidor.Text, 993, True)

            client.Authenticate(txtUsuario.Text, txtPassword.Text)

            Dim inbox = client.Inbox

            inbox.Open(FolderAccess.ReadOnly)

            lstCorreos.Items.Clear()

            For i = 0 To inbox.Count - 1

                lstCorreos.Items.Add(inbox.GetMessage(i).Subject)

            Next

            client.Disconnect(True)

        End Using

    End Sub

End Class
```

Programa 190. Cliente de correo con adjuntos
Controles:

- TextBox → Name: `txtPara`
- TextBox → Name: `txtAsunto`
- TextBox → Name: `txtCuerpo`, Multiline: True
- Button → Name: `btnAdjuntar`, Text: "Adjuntar Archivo"
- Button → Name: `btnEnviar`, Text: "Enviar"
- OpenFileDialog → Name: `OpenFileDialog1`
- ListBox → Name: `lstAdjuntos`

Código:

```
Imports System.Net.Mail


Public Class Form1
    Dim archivosAdjuntos As New List(Of String)


    Private Sub btnAdjuntar_Click(sender As Object, e As EventArgs) Handles
btnAdjuntar.Click
        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
            archivosAdjuntos.Add(OpenFileDialog1.FileName)
            lstAdjuntos.Items.Add(IO.Path.GetFileName(OpenFileDialog1.FileName))
        End If
    End Sub


    Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles
btnEnviar.Click
        Dim mail As New MailMessage()
        mail.To.Add(txtPara.Text)
        mail.Subject = txtAsunto.Text
        mail.Body = txtCuerpo.Text
        For Each file In archivosAdjuntos
            mail.Attachments.Add(New Attachment(file))
        Next
        MessageBox.Show("Correo preparado con adjuntos (SMTP necesario para
envío real)")
    End Sub
End Class
```

Programa 200. Cliente de correo con múltiples cuentas
Controles:

- ComboBox → Name: `cmbCuentas` (Items: "Cuenta1", "Cuenta2")
- TextBox → Name: `txtPara`
- TextBox → Name: `txtAsunto`
- TextBox → Name: `txtCuerpo`, Multiline: True
- Button → Name: `btnEnviar`, Text: "Enviar"

Código conceptual:

```
Public Class Form1

    Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles
btnEnviar.Click

        Dim cuentaSeleccionada As String = cmbCuentas.SelectedItem

        MessageBox.Show("Correo enviado desde " & cuentaSeleccionada & " a " &
txtPara.Text)

    End Sub

End Class
```

Programa 201. Cliente de correo con filtros de mensajes
Controles:

- TextBox → Name: `txtFiltro`
- Button → Name: `btnAplicarFiltro`, Text: "Aplicar Filtro"
- ListBox → Name: `lstCorreos`

Código conceptual:

```
Private Sub btnAplicarFiltro_Click(sender As Object, e As EventArgs) Handles
btnAplicarFiltro.Click

    Dim filtro As String = txtFiltro.Text.ToLower()

    For Each item As String In lstCorreos.Items

        If item.ToLower().Contains(filtro) Then

            MessageBox.Show("Correo filtrado: " & item)

        End If

    Next

End Sub
```

Programa 202. Cliente de correo con carpetas personalizadas
Controles:

- TreeView → Name: `treeCarpetas`
- ListBox → Name: `lstCorreos`
- Button → Name: `btnCrearCarpeta`, Text: "Crear Carpeta"
- TextBox → Name: `txtNuevaCarpeta`

Código:

```
Private Sub btnCrearCarpeta_Click(sender As Object, e As EventArgs) Handles
btnCrearCarpeta.Click

    treeCarpetas.Nodes.Add(txtNuevaCarpeta.Text)

    txtNuevaCarpeta.Clear()

End Sub
```

Programa 203. Cliente de correo con notificaciones
Controles:

- NotifyIcon → Name: `NotifyIcon1`, Icon: `icono.ico`
- Button → Name: `btnNotificar`, Text: "Notificar"

Código:

```
Private Sub btnNotificar_Click(sender As Object, e As EventArgs) Handles
btnNotificar.Click

    NotifyIcon1.Visible = True

    NotifyIcon1.ShowBalloonTip(3000, "Correo", "Tienes un nuevo mensaje",
ToolTipIcon.Info)

End Sub
```

Programa 204. Cliente de correo con contactos
Controles:

- ListBox → Name: `lstContactos`
- TextBox → Name: `txtNombre`
- TextBox → Name: `txtCorreo`
- Button → Name: `btnAgregar`, Text: "Agregar Contacto"

Código:

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    lstContactos.Items.Add(txtNombre.Text & " <" & txtCorreo.Text & ">")
    txtNombre.Clear()
    txtCorreo.Clear()
End Sub
```

Programa 205. Cliente de correo con calendario
Controles:

- MonthCalendar → Name: `calendar`
- Button → Name: `btnSeleccionar`, Text: "Seleccionar Fecha"
- TextBox → Name: `txtFechaSeleccionada`

Código:

```
Private Sub btnSeleccionar_Click(sender As Object, e As EventArgs) Handles
btnSeleccionar.Click
    txtFechaSeleccionada.Text = calendar.SelectionStart.ToShortDateString()
End Sub
```

Programa 206. Cliente de correo con recordatorios
Controles:

- DateTimePicker → Name: `dtpRecordatorio`
- TextBox → Name: `txtMensaje`
- Button → Name: `btnAgregar`, Text: "Agregar Recordatorio"
- ListBox → Name: `lstRecordatorios`

Código:

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles btnAgregar.Click
    lstRecordatorios.Items.Add(dtpRecordatorio.Value.ToString() & " - " & txtMensaje.Text)
    txtMensaje.Clear()
End Sub
```

Programa 207. Cliente de correo con sincronización
Controles:

- Button → Name: `btnSincronizar`, Text: "Sincronizar"
- ListBox → Name: `lstCorreos`

Código conceptual:

```
Private Sub btnSincronizar_Click(sender As Object, e As EventArgs) Handles btnSincronizar.Click
    MessageBox.Show("Sincronización de correos iniciada (implementación real requiere servidor)")
End Sub
```

Programa 208. Cliente de correo con cifrado
Controles:

- TextBox → Name: `txtCuerpo`, Multiline: True
- Button → Name: `btnCifrar`, Text: "Cifrar"
- TextBox → Name: `txtCifrado`, Multiline: True

Código simple con Base64:

```
Imports System.Text


Private Sub btnCifrar_Click(sender As Object, e As EventArgs) Handles
btnCifrar.Click

    txtCifrado.Text =
Convert.ToBase64String(Encoding.UTF8.GetBytes(txtCuerpo.Text))

End Sub
```

Programa 209. Cliente de correo con firmas personalizadas
Controles:

- TextBox → Name: `txtCuerpo`, Multiline: True
- TextBox → Name: `txtFirma`
- Button → Name: `btnEnviar`, Text: "Enviar"

Código:

```
Private Sub btnEnviar_Click(sender As Object, e As EventArgs) Handles
btnEnviar.Click

    Dim mensajeFinal As String = txtCuerpo.Text & Environment.NewLine &
txtFirma.Text

    MessageBox.Show("Correo listo para enviar: " & Environment.NewLine &
mensajeFinal)

End Sub
```

Programa 210. Sistema de login con logs de actividad
Controles:

- TextBox → Name: `txtUsuario`
- TextBox → Name: `txtContraseña`, PasswordChar: *
- Button → Name: `btnLogin`, Text: "Iniciar Sesión"

Código:

```
Imports System.IO


Public Class Form1
    Dim logArchivo As String = "log_actividad.txt"


    Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
        If txtUsuario.Text <> "" And txtContraseña.Text <> "" Then
            File.AppendAllText(logArchivo, DateTime.Now.ToString() & " - " &
txtUsuario.Text & " inició sesión" & Environment.NewLine)
            MessageBox.Show("Bienvenido " & txtUsuario.Text)
        Else
            MessageBox.Show("Ingrese usuario y contraseña")
        End If
    End Sub
End Class
```

Programa 211. Sistema de login con auditoría
Controles:

- TextBox → Name: `txtUsuario`
- TextBox → Name: `txtContraseña`, PasswordChar: *
- Button → Name: `btnLogin`, Text: "Iniciar Sesión"
- ListBox → Name: `lstAuditoria`

Código:

```
Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
    Dim auditoria As String = DateTime.Now.ToString() & " - Usuario: " &
txtUsuario.Text & " intentó iniciar sesión"
    lstAuditoria.Items.Add(auditoria)
    MessageBox.Show("Intento registrado")
```

```
End Sub
```

Programa 212. Sistema de login con autenticación externa
Controles:

- TextBox → Name: `txtUsuario`
- TextBox → Name: `txtContraseña`, PasswordChar: *
- Button → Name: `btnLogin`, Text: "Iniciar Sesión"

Código conceptual:

```
Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
    ' Autenticación externa simulada
    If txtUsuario.Text = "admin" And txtContraseña.Text = "1234" Then
        MessageBox.Show("Autenticación exitosa (externa)")
    Else
        MessageBox.Show("Autenticación fallida")
    End If
End Sub
```

Programa 213. Sistema de login con validación de IP
Controles:

- TextBox → Name: `txtUsuario`
- TextBox → Name: `txtContraseña`, PasswordChar: *
- Button → Name: `btnLogin`, Text: "Iniciar Sesión"

Código conceptual:

```
Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
    Dim ipActual As String = "192.168.1.100" ' Simulada
    Dim ipPermitida As String = "192.168.1.100"
    If ipActual = ipPermitida Then
        MessageBox.Show("IP válida, acceso permitido")
    Else
        MessageBox.Show("IP no permitida")
    End If
End Sub
```

Programa 214. Sistema de login con tiempo de expiración
Controles:

- TextBox → Name: `txtUsuario`
- TextBox → Name: `txtContraseña`, PasswordChar: *
- Button → Name: `btnLogin`, Text: "Iniciar Sesión"

Código:

```
Dim tiempoExpiracion As DateTime = DateTime.Now.AddMinutes(30)


Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
    If DateTime.Now <= tiempoExpiracion Then
        MessageBox.Show("Sesión iniciada correctamente")
    Else
        MessageBox.Show("Tiempo de sesión expirado")
    End If
End Sub
```

Programa 215. Sistema de login con recordatorio de sesión
Controles:

- Timer → Name: `Timer1`, Interval: 60000 (1 min)
- Label → Name: `lblRecordatorio`
- TextBox → Name: `txtUsuario`
- TextBox → Name: `txtContraseña`, PasswordChar: *
- Button → Name: `btnLogin`, Text: "Iniciar Sesión"

Código:

```
Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles Timer1.Tick
    lblRecordatorio.Text = "Recuerda cerrar sesión si terminas"
End Sub


Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
    Timer1.Start()
    MessageBox.Show("Sesión iniciada")
End Sub
```

Programa 216. Sistema de login con restricción de horarios
Controles:

- TextBox → Name: `txtUsuario`
- TextBox → Name: `txtContraseña`, PasswordChar: *
- Button → Name: `btnLogin`, Text: "Iniciar Sesión"

Código:

```
Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
    Dim horaActual As TimeSpan = DateTime.Now.TimeOfDay
    Dim inicio As TimeSpan = New TimeSpan(8, 0, 0)
    Dim fin As TimeSpan = New TimeSpan(18, 0, 0)
    If horaActual >= inicio And horaActual <= fin Then
        MessageBox.Show("Acceso permitido")
    Else
        MessageBox.Show("Acceso fuera de horario permitido")
    End If
End Sub
```

Programa 217. Sistema de login con acceso remoto
Controles:

- TextBox → Name: `txtUsuario`
- TextBox → Name: `txtContraseña`, PasswordChar: *
- Button → Name: `btnLogin`, Text: "Iniciar Sesión"

Código conceptual:

```
Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
    MessageBox.Show("Acceso remoto verificado (implementación real requiere
servidor)")
End Sub
```

Programa 218. Sistema de login con sincronización de datos
Controles:

- TextBox → Name: `txtUsuario`
- TextBox → Name: `txtContraseña`, PasswordChar: *
- Button → Name: `btnLogin`, Text: "Iniciar Sesión"

Código conceptual:

```
Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click

    MessageBox.Show("Datos sincronizados con servidor al iniciar sesión")

End Sub
```

Programa 219. Sistema de login con cifrado de base de datos
Controles:

- TextBox → Name: `txtUsuario`
- TextBox → Name: `txtContraseña`, PasswordChar: *
- Button → Name: `btnLogin`, Text: "Iniciar Sesión"

Código ejemplo simple con Base64:

```
Imports System.Text


Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click

    Dim contrasenaCifrada As String =
Convert.ToBase64String(Encoding.UTF8.GetBytes(txtContraseña.Text))

    MessageBox.Show("Contraseña cifrada para base de datos: " &
contrasenaCifrada)

End Sub
```

Programa 220. Sistema de login con hash de contraseñas
Controles:

- TextBox → Name: `txtUsuario`
- TextBox → Name: `txtContraseña`, PasswordChar: *
- Button → Name: `btnLogin`, Text: "Iniciar Sesión"

Código usando SHA256:

```
Imports System.Security.Cryptography
Imports System.Text


Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
    Dim sha As SHA256 = SHA256.Create()
    Dim bytes As Byte() = Encoding.UTF8.GetBytes(txtContraseña.Text)
    Dim hash As Byte() = sha.ComputeHash(bytes)
    Dim hashString As String = BitConverter.ToString(hash).Replace("-", "")
    MessageBox.Show("Hash de contraseña: " & hashString)
End Sub
```

Programa 221. Sistema de inventario básico
Controles:

- TextBox → Name: `txtProducto`
- TextBox → Name: `txtCantidad`
- Button → Name: `btnAgregar`, Text: "Agregar"
- ListBox → Name: `lstInventario`

Código:

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    lstInventario.Items.Add(txtProducto.Text & " - Cantidad: " &
txtCantidad.Text)
    txtProducto.Clear()
    txtCantidad.Clear()
End Sub
```

Programa 222. Sistema de inventario con base de datos
Controles:

- TextBox → Name: `txtProducto`
- TextBox → Name: `txtCantidad`
- Button → Name: `btnAgregar`, Text: "Agregar"
- DataGridView → Name: `dgvInventario`

Código conceptual usando DataTable:

```
Dim dt As New DataTable()


Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    dt.Columns.Add("Producto")
    dt.Columns.Add("Cantidad")
    dgvInventario.DataSource = dt
End Sub


Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    dt.Rows.Add(txtProducto.Text, txtCantidad.Text)
    txtProducto.Clear()
    txtCantidad.Clear()
End Sub
```

Programa 223. Sistema de inventario con facturación
Controles:

- TextBox → Name: `txtProducto`
- TextBox → Name: `txtCantidad`
- TextBox → Name: `txtPrecio`
- Button → Name: `btnAgregar`, Text: "Agregar"
- ListBox → Name: `lstFactura`

Código:

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click

    Dim total As Double = Val(txtCantidad.Text) * Val(txtPrecio.Text)

    lstFactura.Items.Add(txtProducto.Text & " - Cantidad: " & txtCantidad.Text &
" - Total: " & total)

    txtProducto.Clear()

    txtCantidad.Clear()

    txtPrecio.Clear()

End Sub
```

Programa 224. Sistema de inventario con reportes
Controles:

- Button → Name: `btnReporte`, Text: "Generar Reporte"
- ListBox → Name: `lstInventario`

Código conceptual:

```
Private Sub btnReporte_Click(sender As Object, e As EventArgs) Handles
btnReporte.Click

    Dim reporte As String = "Reporte Inventario:" & Environment.NewLine

    For Each item In lstInventario.Items

        reporte &= item.ToString() & Environment.NewLine

    Next

    MessageBox.Show(reporte)

End Sub
```

Programa 225. Sistema de inventario con gráficos
Controles:

- Chart → Name: `chartInventario`, ChartType: Column
- Button → Name: `btnGraficar`, Text: "Mostrar Gráfico"

Código conceptual:

```
Private Sub btnGraficar_Click(sender As Object, e As EventArgs) Handles
btnGraficar.Click
    chartInventario.Series.Clear()
    Dim serie As New DataVisualization.Charting.Series("Inventario")
    serie.ChartType = DataVisualization.Charting.SeriesChartType.Column
    serie.Points.AddXY("Producto A", 10)
    serie.Points.AddXY("Producto B", 15)
    chartInventario.Series.Add(serie)
End Sub
```

Programa 226. Sistema de inventario con exportación Excel
Controles:

- Button → Name: `btnExportar`, Text: "Exportar a Excel"

Código conceptual:

```
Imports Microsoft.Office.Interop.Excel


Private Sub btnExportar_Click(sender As Object, e As EventArgs) Handles
btnExportar.Click
    MessageBox.Show("Datos exportados a Excel (requiere referencia a Microsoft
Excel)")
End Sub
```

Programa 227. Sistema de inventario con importación Excel
Controles:

- Button → Name: `btnImportar`, Text: "Importar desde Excel"

Código conceptual:

```
Imports Microsoft.Office.Interop.Excel


Private Sub btnImportar_Click(sender As Object, e As EventArgs) Handles
btnImportar.Click
    MessageBox.Show("Datos importados desde Excel (requiere referencia a
Microsoft Excel)")
End Sub
```

Programa 228. Sistema de inventario con códigos de barra
Controles:

- TextBox → Name: `txtCodigo`
- Button → Name: `btnAgregar`, Text: "Agregar Código"
- ListBox → Name: `lstCodigos`

Código:

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click

    lstCodigos.Items.Add(txtCodigo.Text)

    txtCodigo.Clear()

End Sub
```

Programa 229. Sistema de inventario con múltiples usuarios
Controles:

- ComboBox → Name: `cmbUsuarios` (Items: "Admin", "Vendedor")
- TextBox → Name: `txtProducto`
- TextBox → Name: `txtCantidad`
- Button → Name: `btnAgregar`, Text: "Agregar"

Código:

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click

    Dim usuario As String = cmbUsuarios.SelectedItem

    MessageBox.Show("Usuario: " & usuario & " agregó " & txtProducto.Text & "
Cantidad: " & txtCantidad.Text)

End Sub
```

Programa 230. Sistema de inventario con permisos
Controles:

- ComboBox → Name: `cmbRoles` (Items: "Admin", "Vendedor")
- Button → Name: `btnEliminar`, Text: "Eliminar Producto"

Código:

```
Private Sub btnEliminar_Click(sender As Object, e As EventArgs) Handles
btnEliminar.Click
    If cmbRoles.SelectedItem = "Admin" Then
        MessageBox.Show("Producto eliminado")
    Else
        MessageBox.Show("No tiene permisos para eliminar")
    End If
End Sub
```

Programa 231. Sistema de inventario con auditoría
Controles:

- TextBox → Name: `txtProducto`
- TextBox → Name: `txtCantidad`
- Button → Name: `btnAgregar`, Text: "Agregar"
- ListBox → Name: `lstInventario`
- ListBox → Name: `lstAuditoria`

Código:

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    lstInventario.Items.Add(txtProducto.Text & " - Cantidad: " &
txtCantidad.Text)
    lstAuditoria.Items.Add(DateTime.Now.ToString() & " - Producto agregado: " &
txtProducto.Text)
    txtProducto.Clear()
    txtCantidad.Clear()
End Sub
```

Programa 232. Sistema de inventario con estadísticas
Controles:

- Button → Name: `btnEstadisticas`, Text: "Mostrar Estadísticas"
- ListBox → Name: `lstInventario`

Código conceptual:

```
Private Sub btnEstadisticas_Click(sender As Object, e As EventArgs) Handles
btnEstadisticas.Click
    Dim totalProductos As Integer = lstInventario.Items.Count
    MessageBox.Show("Total de productos en inventario: " & totalProductos)
End Sub
```

Programa 233. Sistema de inventario con gráficos dinámicos
Controles:

- Chart → Name: `chartInventario`, ChartType: Column
- Button → Name: `btnActualizarGrafico`, Text: "Actualizar Gráfico"
- ListBox → Name: `lstInventario`

Código conceptual:

```
Private Sub btnActualizarGrafico_Click(sender As Object, e As EventArgs) Handles
btnActualizarGrafico.Click
    chartInventario.Series.Clear()
    Dim serie As New DataVisualization.Charting.Series("Inventario")
    serie.ChartType = DataVisualization.Charting.SeriesChartType.Column
    For Each item In lstInventario.Items
        serie.Points.AddXY(item.ToString(), 1)
    Next
    chartInventario.Series.Add(serie)
End Sub
```

Programa 234. Sistema de inventario con múltiples almacenes
Controles:

- ComboBox → Name: `cmbAlmacenes` (Items: "Almacén 1", "Almacén 2")
- TextBox → Name: `txtProducto`
- TextBox → Name: `txtCantidad`
- Button → Name: `btnAgregar`, Text: "Agregar"
- ListBox → Name: `lstInventario`

Código:

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click

    Dim almacen As String = cmbAlmacenes.SelectedItem

    lstInventario.Items.Add(almacen & " - " & txtProducto.Text & " - Cantidad: "
& txtCantidad.Text)

    txtProducto.Clear()

    txtCantidad.Clear()

End Sub
```

Programa 235. Sistema de inventario con conexión remota
Controles:

- Button → Name: `btnConectar`, Text: "Conectar a Servidor"

Código conceptual:

```
Private Sub btnConectar_Click(sender As Object, e As EventArgs) Handles
btnConectar.Click

    MessageBox.Show("Conexión remota establecida (simulada)")

End Sub
```

Programa 236. Sistema de inventario con backup automático
Controles:

- Button → Name: `btnBackup`, Text: "Realizar Backup"

Código conceptual:

```
Private Sub btnBackup_Click(sender As Object, e As EventArgs) Handles
btnBackup.Click

    MessageBox.Show("Backup automático realizado correctamente")

End Sub
```

Programa 237. Sistema de inventario con restauración de datos
Controles:

- Button → Name: `btnRestaurar`, Text: "Restaurar Backup"

Código conceptual:

```
Private Sub btnRestaurar_Click(sender As Object, e As EventArgs) Handles
btnRestaurar.Click

    MessageBox.Show("Datos restaurados desde backup")

End Sub
```

Programa 238. Sistema de inventario con notificaciones
Controles:

- NotifyIcon → Name: `NotifyIcon1`, Icon: `icono.ico`
- Button → Name: `btnNotificar`, Text: "Notificar Stock Bajo"

Código:

```
Private Sub btnNotificar_Click(sender As Object, e As EventArgs) Handles
btnNotificar.Click

    NotifyIcon1.Visible = True

    NotifyIcon1.ShowBalloonTip(3000, "Inventario", "Alerta: Stock bajo en
algunos productos", ToolTipIcon.Warning)

End Sub
```

Programa 239. Sistema de inventario con control de lotes
Controles:

- TextBox → Name: `txtProducto`
- TextBox → Name: `txtLote`
- TextBox → Name: `txtCantidad`
- Button → Name: `btnAgregar`, Text: "Agregar Lote"
- ListBox → Name: `lstInventario`

Código:

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click

    lstInventario.Items.Add("Producto: " & txtProducto.Text & " - Lote: " &
txtLote.Text & " - Cantidad: " & txtCantidad.Text)

    txtProducto.Clear()

    txtLote.Clear()

    txtCantidad.Clear()

End Sub
```

Programa 240. Sistema de inventario con caducidades
Controles:

- TextBox → Name: `txtProducto`
- DateTimePicker → Name: `dtpCaducidad`
- TextBox → Name: `txtCantidad`
- Button → Name: `btnAgregar`, Text: "Agregar Producto"
- ListBox → Name: `lstInventario`

Código:

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click

    lstInventario.Items.Add("Producto: " & txtProducto.Text & " - Cantidad: " &
txtCantidad.Text & " - Caduca: " & dtpCaducidad.Value.ToShortDateString())

    txtProducto.Clear()

    txtCantidad.Clear()

End Sub
```

Programa 231. Sistema de inventario con auditoría
Controles:

- TextBox → Name: `txtProducto`
- TextBox → Name: `txtCantidad`
- Button → Name: `btnAgregar`, Text: "Agregar"
- ListBox → Name: `lstInventario`
- ListBox → Name: `lstAuditoria`

Código:

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click

    lstInventario.Items.Add(txtProducto.Text & " - Cantidad: " &
txtCantidad.Text)

    lstAuditoria.Items.Add(DateTime.Now.ToString() & " - Producto agregado: " &
txtProducto.Text)

    txtProducto.Clear()

    txtCantidad.Clear()

End Sub
```

Programa 232. Sistema de inventario con estadísticas
Controles:

- Button → Name: `btnEstadisticas`, Text: "Mostrar Estadísticas"
- ListBox → Name: `lstInventario`

Código conceptual:

```
Private Sub btnEstadisticas_Click(sender As Object, e As EventArgs) Handles
btnEstadisticas.Click
    Dim totalProductos As Integer = lstInventario.Items.Count
    MessageBox.Show("Total de productos en inventario: " & totalProductos)
End Sub
```

Programa 233. Sistema de inventario con gráficos dinámicos
Controles:

- Chart → Name: `chartInventario`, ChartType: Column
- Button → Name: `btnActualizarGrafico`, Text: "Actualizar Gráfico"
- ListBox → Name: `lstInventario`

Código conceptual:

```
Private Sub btnActualizarGrafico_Click(sender As Object, e As EventArgs) Handles
btnActualizarGrafico.Click
    chartInventario.Series.Clear()
    Dim serie As New DataVisualization.Charting.Series("Inventario")
    serie.ChartType = DataVisualization.Charting.SeriesChartType.Column
    For Each item In lstInventario.Items
        serie.Points.AddXY(item.ToString(), 1)
    Next
    chartInventario.Series.Add(serie)
End Sub
```

Programa 234. Sistema de inventario con múltiples almacenes
Controles:

- ComboBox → Name: `cmbAlmacenes` (Items: "Almacén 1", "Almacén 2")
- TextBox → Name: `txtProducto`
- TextBox → Name: `txtCantidad`
- Button → Name: `btnAgregar`, Text: "Agregar"
- ListBox → Name: `lstInventario`

Código:

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click

    Dim almacen As String = cmbAlmacenes.SelectedItem

    lstInventario.Items.Add(almacen & " - " & txtProducto.Text & " - Cantidad: "
& txtCantidad.Text)

    txtProducto.Clear()

    txtCantidad.Clear()

End Sub
```

Programa 235. Sistema de inventario con conexión remota
Controles:

- Button → Name: `btnConectar`, Text: "Conectar a Servidor"

Código conceptual:

```
Private Sub btnConectar_Click(sender As Object, e As EventArgs) Handles
btnConectar.Click

    MessageBox.Show("Conexión remota establecida (simulada)")

End Sub
```

Programa 236. Sistema de inventario con backup automático
Controles:

- Button → Name: `btnBackup`, Text: "Realizar Backup"

Código conceptual:

```
Private Sub btnBackup_Click(sender As Object, e As EventArgs) Handles
btnBackup.Click

    MessageBox.Show("Backup automático realizado correctamente")

End Sub
```

Programa 237. Sistema de inventario con restauración de datos
Controles:

- Button → Name: `btnRestaurar`, Text: "Restaurar Backup"

Código conceptual:

```
Private Sub btnRestaurar_Click(sender As Object, e As EventArgs) Handles
btnRestaurar.Click

    MessageBox.Show("Datos restaurados desde backup")

End Sub
```

Programa 238. Sistema de inventario con notificaciones
Controles:

- NotifyIcon → Name: `NotifyIcon1`, Icon: `icono.ico`
- Button → Name: `btnNotificar`, Text: "Notificar Stock Bajo"

Código:

```
Private Sub btnNotificar_Click(sender As Object, e As EventArgs) Handles
btnNotificar.Click

    NotifyIcon1.Visible = True

    NotifyIcon1.ShowBalloonTip(3000, "Inventario", "Alerta: Stock bajo en
algunos productos", ToolTipIcon.Warning)

End Sub
```

Programa 239. Sistema de inventario con control de lotes
Controles:

- TextBox → Name: `txtProducto`
- TextBox → Name: `txtLote`
- TextBox → Name: `txtCantidad`
- Button → Name: `btnAgregar`, Text: "Agregar Lote"
- ListBox → Name: `lstInventario`

Código:

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click

    lstInventario.Items.Add("Producto: " & txtProducto.Text & " - Lote: " &
txtLote.Text & " - Cantidad: " & txtCantidad.Text)

    txtProducto.Clear()

    txtLote.Clear()

    txtCantidad.Clear()

End Sub
```

Programa 240. Sistema de inventario con caducidades
Controles:

- TextBox → Name: `txtProducto`
- DateTimePicker → Name: `dtpCaducidad`
- TextBox → Name: `txtCantidad`
- Button → Name: `btnAgregar`, Text: "Agregar Producto"
- ListBox → Name: `lstInventario`

Código:

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click

    lstInventario.Items.Add("Producto: " & txtProducto.Text & " - Cantidad: " &
txtCantidad.Text & " - Caduca: " & dtpCaducidad.Value.ToShortDateString())

    txtProducto.Clear()

    txtCantidad.Clear()

End Sub
```

Programa 241. Sistema de ventas básico
Controles:

- TextBox → Name: `txtProducto`
- TextBox → Name: `txtCantidad`
- Button → Name: `btnAgregar`, Text: "Agregar Venta"
- ListBox → Name: `lstVentas`

Código:

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click

    lstVentas.Items.Add(txtProducto.Text & " - Cantidad: " & txtCantidad.Text)

    txtProducto.Clear()

    txtCantidad.Clear()

End Sub
```

Programa 242. Sistema de ventas con base de datos
Controles:

- TextBox → Name: `txtProducto`
- TextBox → Name: `txtCantidad`
- Button → Name: `btnAgregar`, Text: "Agregar Venta"
- DataGridView → Name: `dgvVentas`

Código conceptual usando DataTable:

```
Dim dt As New DataTable()


Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    dt.Columns.Add("Producto")
    dt.Columns.Add("Cantidad")
    dgvVentas.DataSource = dt
End Sub


Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    dt.Rows.Add(txtProducto.Text, txtCantidad.Text)
    txtProducto.Clear()
    txtCantidad.Clear()
End Sub
```

Programa 243. Sistema de ventas con clientes
Controles:

- ComboBox → Name: `cmbClientes` (Items: "Cliente 1", "Cliente 2")
- TextBox → Name: `txtProducto`
- TextBox → Name: `txtCantidad`
- Button → Name: `btnAgregar`, Text: "Agregar Venta"
- ListBox → Name: `lstVentas`

Código:

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click

    lstVentas.Items.Add("Cliente: " & cmbClientes.SelectedItem & " - Producto: "
& txtProducto.Text & " - Cantidad: " & txtCantidad.Text)

    txtProducto.Clear()

    txtCantidad.Clear()

End Sub
```

Programa 244. Sistema de ventas con facturación
Controles:

- TextBox → Name: `txtProducto`
- TextBox → Name: `txtCantidad`
- TextBox → Name: `txtPrecio`
- Button → Name: `btnAgregar`, Text: "Agregar Factura"
- ListBox → Name: `lstFactura`

Código:

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click

    Dim total As Double = Val(txtCantidad.Text) * Val(txtPrecio.Text)

    lstFactura.Items.Add(txtProducto.Text & " - Cantidad: " & txtCantidad.Text &
" - Total: " & total)

    txtProducto.Clear()

    txtCantidad.Clear()

    txtPrecio.Clear()

End Sub
```

Programa 245. Sistema de ventas con reportes
Controles:

- Button → Name: `btnReporte`, Text: "Generar Reporte"
- ListBox → Name: `lstVentas`

Código:

```
Private Sub btnReporte_Click(sender As Object, e As EventArgs) Handles
btnReporte.Click

    Dim reporte As String = "Reporte de Ventas:" & Environment.NewLine

    For Each item In lstVentas.Items

        reporte &= item.ToString() & Environment.NewLine

    Next

    MessageBox.Show(reporte)

End Sub
```

Programa 246. Sistema de ventas con estadísticas
Controles:

- Button → Name: `btnEstadisticas`, Text: "Mostrar Estadísticas"
- ListBox → Name: `lstVentas`

Código conceptual:

```
Private Sub btnEstadisticas_Click(sender As Object, e As EventArgs) Handles
btnEstadisticas.Click

    Dim totalVentas As Integer = lstVentas.Items.Count

    MessageBox.Show("Total de ventas registradas: " & totalVentas)

End Sub
```

Programa 247. Sistema de ventas con gráficos
Controles:

- Chart → Name: `chartVentas`, ChartType: Column
- Button → Name: `btnGraficar`, Text: "Mostrar Gráfico"
- ListBox → Name: `lstVentas`

Código conceptual:

```
Private Sub btnGraficar_Click(sender As Object, e As EventArgs) Handles
btnGraficar.Click
    chartVentas.Series.Clear()
    Dim serie As New DataVisualization.Charting.Series("Ventas")
    serie.ChartType = DataVisualization.Charting.SeriesChartType.Column
    For Each item In lstVentas.Items
        serie.Points.AddXY(item.ToString(), 1)
    Next
    chartVentas.Series.Add(serie)
End Sub
```

Programa 248. Sistema de ventas con exportación Excel
Controles:

- Button → Name: `btnExportar`, Text: "Exportar a Excel"

Código conceptual:

```
Imports Microsoft.Office.Interop.Excel


Private Sub btnExportar_Click(sender As Object, e As EventArgs) Handles
btnExportar.Click
    MessageBox.Show("Ventas exportadas a Excel (requiere referencia a Microsoft
Excel)")
End Sub
```

Programa 249. Sistema de ventas con importación Excel
Controles:

- Button → Name: `btnImportar`, Text: "Importar desde Excel"

Código conceptual:

```
Imports Microsoft.Office.Interop.Excel


Private Sub btnImportar_Click(sender As Object, e As EventArgs) Handles
btnImportar.Click

    MessageBox.Show("Ventas importadas desde Excel (requiere referencia a
Microsoft Excel)")

End Sub
```

Programa 250. Sistema de ventas con múltiples usuarios
Controles:

- ComboBox → Name: `cmbUsuarios` (Items: "Admin", "Vendedor")
- TextBox → Name: `txtProducto`
- TextBox → Name: `txtCantidad`
- Button → Name: `btnAgregar`, Text: "Agregar Venta"

Código:

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click

    Dim usuario As String = cmbUsuarios.SelectedItem

    MessageBox.Show("Usuario: " & usuario & " agregó Producto: " &
txtProducto.Text & " Cantidad: " & txtCantidad.Text)

End Sub
```

Programa 251. Sistema de ventas con permisos
Controles:

- ComboBox → Name: `cmbRoles` (Items: "Admin", "Vendedor")
- Button → Name: `btnEliminar`, Text: "Eliminar Venta"

Código:

```
Private Sub btnEliminar_Click(sender As Object, e As EventArgs) Handles
btnEliminar.Click
    If cmbRoles.SelectedItem = "Admin" Then
        MessageBox.Show("Venta eliminada")
    Else
        MessageBox.Show("No tiene permisos para eliminar")
    End If
End Sub
```

Programa 252. Sistema de ventas con auditoría
Controles:

- TextBox → Name: `txtProducto`
- TextBox → Name: `txtCantidad`
- Button → Name: `btnAgregar`, Text: "Agregar Venta"
- ListBox → Name: `lstVentas`
- ListBox → Name: `lstAuditoria`

Código:

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    lstVentas.Items.Add(txtProducto.Text & " - Cantidad: " & txtCantidad.Text)
    lstAuditoria.Items.Add(DateTime.Now.ToString() & " - Venta agregada: " &
txtProducto.Text)
    txtProducto.Clear()
    txtCantidad.Clear()
End Sub
```

Programa 253. Sistema de ventas con conexión remota
Controles:

- Button → Name: `btnConectar`, Text: "Conectar a Servidor"

Código conceptual:

```
Private Sub btnConectar_Click(sender As Object, e As EventArgs) Handles
btnConectar.Click

    MessageBox.Show("Conexión remota establecida (simulada)")

End Sub
```

Programa 254. Sistema de ventas con backup automático
Controles:

- Button → Name: `btnBackup`, Text: "Realizar Backup"

Código conceptual:

```
Private Sub btnBackup_Click(sender As Object, e As EventArgs) Handles
btnBackup.Click

    MessageBox.Show("Backup automático de ventas realizado correctamente")

End Sub
```

Programa 255. Sistema de ventas con restauración de datos
Controles:

- Button → Name: `btnRestaurar`, Text: "Restaurar Backup"

Código conceptual:

```
Private Sub btnRestaurar_Click(sender As Object, e As EventArgs) Handles
btnRestaurar.Click

    MessageBox.Show("Ventas restauradas desde backup")

End Sub
```

Programa 256. Sistema de ventas con notificaciones
Controles:

- NotifyIcon → Name: `NotifyIcon1`, Icon: `icono.ico`
- Button → Name: `btnNotificar`, Text: "Notificar Venta"

Código:

```
Private Sub btnNotificar_Click(sender As Object, e As EventArgs) Handles
btnNotificar.Click

    NotifyIcon1.Visible = True

    NotifyIcon1.ShowBalloonTip(3000, "Ventas", "Nueva venta registrada",
ToolTipIcon.Info)

End Sub
```

Programa 257. Sistema de ventas con control de stock
Controles:

- TextBox → Name: `txtProducto`
- TextBox → Name: `txtCantidad`
- Button → Name: `btnAgregar`, Text: "Agregar Venta"
- Dictionary(Of String, Integer) → `stock`

Código:

```
Dim stock As New Dictionary(Of String, Integer) From {{"ProductoA", 10},
{"ProductoB", 5}}


Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click

    Dim producto As String = txtProducto.Text

    Dim cantidad As Integer = Val(txtCantidad.Text)

    If stock.ContainsKey(producto) AndAlso stock(producto) >= cantidad Then

        stock(producto) -= cantidad

        MessageBox.Show("Venta realizada. Stock restante: " & stock(producto))

    Else

        MessageBox.Show("Stock insuficiente")

    End If

End Sub
```

Programa 258. Sistema de ventas con caducidades
Controles:

- TextBox → Name: `txtProducto`
- DateTimePicker → Name: `dtpCaducidad`
- TextBox → Name: `txtCantidad`
- Button → Name: `btnAgregar`, Text: "Agregar Venta"
- ListBox → Name: `lstVentas`

Código:

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click

    lstVentas.Items.Add("Producto: " & txtProducto.Text & " - Cantidad: " &
txtCantidad.Text & " - Caduca: " & dtpCaducidad.Value.ToShortDateString())

    txtProducto.Clear()

    txtCantidad.Clear()

End Sub
```

Programa 259. Sistema de ventas con código de barras
Controles:

- TextBox → Name: `txtCodigo`
- Button → Name: `btnAgregar`, Text: "Agregar Venta"
- ListBox → Name: `lstVentas`

Código:

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click

    lstVentas.Items.Add("Código: " & txtCodigo.Text)

    txtCodigo.Clear()

End Sub
```

Programa 260. Sistema de ventas con descuentos
Controles:

- TextBox → Name: `txtProducto`
- TextBox → Name: `txtPrecio`
- TextBox → Name: `txtDescuento`
- Button → Name: `btnAgregar`, Text: "Agregar Venta"
- ListBox → Name: `lstVentas`

Código:

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click

    Dim precio As Double = Val(txtPrecio.Text)

    Dim descuento As Double = Val(txtDescuento.Text)

    Dim total As Double = precio - (precio * descuento / 100)

    lstVentas.Items.Add(txtProducto.Text & " - Precio: " & precio & " -
Descuento: " & descuento & "% - Total: " & total)

    txtProducto.Clear()

    txtPrecio.Clear()

    txtDescuento.Clear()

End Sub
```

Programa 261. Sistema escolar básico
Controles:

- TextBox → Name: `txtAlumno`
- TextBox → Name: `txtMateria`
- Button → Name: `btnAgregar`, Text: "Agregar Registro"
- ListBox → Name: `lstRegistros`

Código:

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click

    lstRegistros.Items.Add("Alumno: " & txtAlumno.Text & " - Materia: " &
txtMateria.Text)

    txtAlumno.Clear()

    txtMateria.Clear()

End Sub
```

Programa 262. Sistema escolar con base de datos
Controles:

- TextBox → Name: `txtAlumno`
- TextBox → Name: `txtMateria`
- Button → Name: `btnAgregar`, Text: "Agregar Registro"
- DataGridView → Name: `dgvRegistros`

Código conceptual con DataTable:

```
Dim dt As New DataTable()


Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    dt.Columns.Add("Alumno")
    dt.Columns.Add("Materia")
    dgvRegistros.DataSource = dt
End Sub


Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles
btnAgregar.Click
    dt.Rows.Add(txtAlumno.Text, txtMateria.Text)
    txtAlumno.Clear()
    txtMateria.Clear()
End Sub
```

Programa 263. Sistema escolar con registro de alumnos
Controles:

- TextBox → Name: `txtNombreAlumno`
- TextBox → Name: `txtEdad`
- Button → Name: `btnAgregarAlumno`, Text: "Agregar Alumno"
- ListBox → Name: `lstAlumnos`

Código:

```
Private Sub btnAgregarAlumno_Click(sender As Object, e As EventArgs) Handles
btnAgregarAlumno.Click
    lstAlumnos.Items.Add("Nombre: " & txtNombreAlumno.Text & " - Edad: " &
txtEdad.Text)
    txtNombreAlumno.Clear()
    txtEdad.Clear()
End Sub
```

Programa 264. Sistema escolar con registro de profesores
Controles:

- TextBox → Name: `txtNombreProfesor`
- TextBox → Name: `txtMateria`
- Button → Name: `btnAgregarProfesor`, Text: "Agregar Profesor"
- ListBox → Name: `lstProfesores`

Código:

```
Private Sub btnAgregarProfesor_Click(sender As Object, e As EventArgs) Handles
btnAgregarProfesor.Click

    lstProfesores.Items.Add("Profesor: " & txtNombreProfesor.Text & " - Materia:
" & txtMateria.Text)

    txtNombreProfesor.Clear()

    txtMateria.Clear()

End Sub
```

Programa 265. Sistema escolar con registro de materias
Controles:

- TextBox → Name: `txtMateria`
- Button → Name: `btnAgregarMateria`, Text: "Agregar Materia"
- ListBox → Name: `lstMaterias`

Código:

```
Private Sub btnAgregarMateria_Click(sender As Object, e As EventArgs) Handles
btnAgregarMateria.Click

    lstMaterias.Items.Add(txtMateria.Text)

    txtMateria.Clear()

End Sub
```

Programa 266. Sistema escolar con registro de calificaciones
Controles:

- TextBox → Name: `txtAlumno`
- TextBox → Name: `txtMateria`
- TextBox → Name: `txtCalificacion`
- Button → Name: `btnAgregarCalificacion`, Text: "Agregar Calificación"
- ListBox → Name: `lstCalificaciones`

Código:

```
Private Sub btnAgregarCalificacion_Click(sender As Object, e As EventArgs)
Handles btnAgregarCalificacion.Click

    lstCalificaciones.Items.Add("Alumno: " & txtAlumno.Text & " - Materia: " &
txtMateria.Text & " - Nota: " & txtCalificacion.Text)

    txtAlumno.Clear()

    txtMateria.Clear()

    txtCalificacion.Clear()

End Sub
```

Programa 267. Sistema escolar con reportes
Controles:

- Button → Name: `btnReporte`, Text: "Generar Reporte"
- ListBox → Name: `lstCalificaciones`

Código:

```
Private Sub btnReporte_Click(sender As Object, e As EventArgs) Handles
btnReporte.Click

    Dim reporte As String = "Reporte de Calificaciones:" & Environment.NewLine

    For Each item In lstCalificaciones.Items

        reporte &= item.ToString() & Environment.NewLine

    Next

    MessageBox.Show(reporte)

End Sub
```

Programa 268. Sistema escolar con gráficos
Controles:

- Chart → Name: `chartCalificaciones`, ChartType: Column
- Button → Name: `btnGraficar`, Text: "Mostrar Gráfico"
- ListBox → Name: `lstCalificaciones`

Código conceptual:

```
Private Sub btnGraficar_Click(sender As Object, e As EventArgs) Handles
btnGraficar.Click
    chartCalificaciones.Series.Clear()
    Dim serie As New DataVisualization.Charting.Series("Calificaciones")
    serie.ChartType = DataVisualization.Charting.SeriesChartType.Column
    For Each item In lstCalificaciones.Items
        serie.Points.AddXY(item.ToString(), 1)
    Next
    chartCalificaciones.Series.Add(serie)
End Sub
```

Programa 269. Sistema escolar con exportación Excel
Controles:

- Button → Name: `btnExportar`, Text: "Exportar a Excel"

Código conceptual:

```
Imports Microsoft.Office.Interop.Excel


Private Sub btnExportar_Click(sender As Object, e As EventArgs) Handles
btnExportar.Click
    MessageBox.Show("Calificaciones exportadas a Excel (requiere referencia a
Microsoft Excel)")
End Sub
```

Programa 270. Sistema escolar con importación Excel
Controles:

- Button → Name: `btnImportar`, Text: "Importar desde Excel"

Código conceptual:

```
Imports Microsoft.Office.Interop.Excel


Private Sub btnImportar_Click(sender As Object, e As EventArgs) Handles
btnImportar.Click

    MessageBox.Show("Calificaciones importadas desde Excel (requiere referencia
a Microsoft Excel)")

End Sub
```

Programa 271. Sistema escolar con múltiples usuarios
Controles:

- ComboBox → Name: `cmbUsuarios` (Items: "Admin", "Profesor", "Alumno")
- Button → Name: `btnAcceder`, Text: "Acceder"

Código:

```
Private Sub btnAcceder_Click(sender As Object, e As EventArgs) Handles
btnAcceder.Click

    MessageBox.Show("Usuario: " & cmbUsuarios.SelectedItem & " ha iniciado
sesión")

End Sub
```

Programa 272. Sistema escolar con permisos
Controles:

- ComboBox → Name: `cmbRoles` (Items: "Admin", "Profesor")
- Button → Name: `btnEliminarRegistro`, Text: "Eliminar Registro"

Código:

```
Private Sub btnEliminarRegistro_Click(sender As Object, e As EventArgs) Handles
btnEliminarRegistro.Click

    If cmbRoles.SelectedItem = "Admin" Then

        MessageBox.Show("Registro eliminado")

    Else

        MessageBox.Show("No tiene permisos para eliminar")

    End If

End Sub
```

Programa 273. Sistema escolar con auditoría
Controles:

- TextBox → Name: `txtAlumno`
- Button → Name: `btnAgregarAlumno`, Text: "Agregar Alumno"
- ListBox → Name: `lstAuditoria`

Código:

```
Private Sub btnAgregarAlumno_Click(sender As Object, e As EventArgs) Handles
btnAgregarAlumno.Click

    lstAuditoria.Items.Add(DateTime.Now.ToString() & " - Alumno agregado: " &
txtAlumno.Text)

    txtAlumno.Clear()

End Sub
```

Programa 274. Sistema escolar con conexión remota
Controles:

- Button → Name: `btnConectar`, Text: "Conectar a Servidor"

Código conceptual:

```
Private Sub btnConectar_Click(sender As Object, e As EventArgs) Handles
btnConectar.Click

    MessageBox.Show("Conexión remota establecida (simulada)")

End Sub
```

Programa 275. Sistema escolar con backup automático
Controles:

- Button → Name: `btnBackup`, Text: "Realizar Backup"

Código conceptual:

```
Private Sub btnBackup_Click(sender As Object, e As EventArgs) Handles
btnBackup.Click

    MessageBox.Show("Backup automático de datos escolares realizado
correctamente")

End Sub
```

Programa 276. Sistema escolar con restauración de datos
Controles:

- Button → Name: `btnRestaurar`, Text: "Restaurar Backup"

Código conceptual:

```
Private Sub btnRestaurar_Click(sender As Object, e As EventArgs) Handles
btnRestaurar.Click

    MessageBox.Show("Datos escolares restaurados desde backup")

End Sub
```

Programa 277. Sistema escolar con notificaciones
Controles:

- NotifyIcon → Name: `NotifyIcon1`, Icon: `icono.ico`
- Button → Name: `btnNotificar`, Text: "Notificar Evento"

Código:

```
Private Sub btnNotificar_Click(sender As Object, e As EventArgs) Handles
btnNotificar.Click

    NotifyIcon1.Visible = True

    NotifyIcon1.ShowBalloonTip(3000, "Sistema Escolar", "Nuevo evento
registrado", ToolTipIcon.Info)

End Sub
```

Programa 278. Sistema escolar con estadísticas
Controles:

- ListBox → Name: `lstCalificaciones`
- Button → Name: `btnEstadisticas`, Text: "Mostrar Estadísticas"

Código:

```
Private Sub btnEstadisticas_Click(sender As Object, e As EventArgs) Handles
btnEstadisticas.Click

    Dim totalAlumnos As Integer = lstCalificaciones.Items.Count

    MessageBox.Show("Total de registros: " & totalAlumnos)

End Sub
```

Programa 279. Sistema escolar con boletines
Controles:

- Button → Name: `btnGenerarBoletin`, Text: "Generar Boletín"
- ListBox → Name: `lstCalificaciones`

Código:

```
Private Sub btnGenerarBoletin_Click(sender As Object, e As EventArgs) Handles
btnGenerarBoletin.Click
    Dim boletin As String = "Boletín de Calificaciones:" & Environment.NewLine
    For Each item In lstCalificaciones.Items
        boletin &= item.ToString() & Environment.NewLine
    Next
    MessageBox.Show(boletin)
End Sub
```

Programa 280. Sistema escolar con login de usuarios
Controles:

- TextBox → Name: `txtUsuario`
- TextBox → Name: `txtContraseña`, PasswordChar: "*"
- Button → Name: `btnLogin`, Text: "Login"

Código:

```
Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click
    If txtUsuario.Text = "admin" And txtContraseña.Text = "1234" Then
        MessageBox.Show("Login exitoso")
    Else
        MessageBox.Show("Usuario o contraseña incorrectos")
    End If
End Sub
```

Programa 281. Sistema médico básico
Controles:

- TextBox → Name: `txtPaciente`
- TextBox → Name: `txtDoctor`
- Button → Name: `btnAgregar`, Text: "Agregar Registro"
- ListBox → Name: `lstRegistros`

Código:

```
Private Sub btnAgregar_Click(sender As Object, e As EventArgs) Handles btnAgregar.Click
    lstRegistros.Items.Add("Paciente: " & txtPaciente.Text & " - Doctor: " & txtDoctor.Text)
    txtPaciente.Clear()
    txtDoctor.Clear()
End Sub
```

Programa 282. Sistema médico con registro de pacientes
Controles:

- TextBox → Name: `txtNombrePaciente`
- TextBox → Name: `txtEdad`
- Button → Name: `btnAgregarPaciente`, Text: "Agregar Paciente"
- ListBox → Name: `lstPacientes`

Código:

```
Private Sub btnAgregarPaciente_Click(sender As Object, e As EventArgs) Handles btnAgregarPaciente.Click
    lstPacientes.Items.Add("Nombre: " & txtNombrePaciente.Text & " - Edad: " & txtEdad.Text)
    txtNombrePaciente.Clear()
    txtEdad.Clear()
End Sub
```

Programa 283. Sistema médico con registro de doctores
Controles:

- TextBox → Name: `txtNombreDoctor`
- TextBox → Name: `txtEspecialidad`
- Button → Name: `btnAgregarDoctor`, Text: "Agregar Doctor"
- ListBox → Name: `lstDoctores`

Código:

```
Private Sub btnAgregarDoctor_Click(sender As Object, e As EventArgs) Handles
btnAgregarDoctor.Click

    lstDoctores.Items.Add("Doctor: " & txtNombreDoctor.Text & " - Especialidad:
" & txtEspecialidad.Text)

    txtNombreDoctor.Clear()

    txtEspecialidad.Clear()

End Sub
```

Programa 284. Sistema médico con registro de citas
Controles:

- TextBox → Name: `txtPaciente`
- TextBox → Name: `txtDoctor`
- DateTimePicker → Name: `dtpFecha`
- Button → Name: `btnAgregarCita`, Text: "Agregar Cita"
- ListBox → Name: `lstCitas`

Código:

```
Private Sub btnAgregarCita_Click(sender As Object, e As EventArgs) Handles
btnAgregarCita.Click

    lstCitas.Items.Add("Paciente: " & txtPaciente.Text & " - Doctor: " &
txtDoctor.Text & " - Fecha: " & dtpFecha.Value.ToShortDateString())

    txtPaciente.Clear()

    txtDoctor.Clear()

End Sub
```

Programa 285. Sistema médico con historial clínico
Controles:

- TextBox → Name: `txtPaciente`
- TextBox → Name: `txtDiagnostico`
- Button → Name: `btnAgregarHistorial`, Text: "Agregar Historial"
- ListBox → Name: `lstHistorial`

Código:

```
Private Sub btnAgregarHistorial_Click(sender As Object, e As EventArgs) Handles btnAgregarHistorial.Click
    lstHistorial.Items.Add("Paciente: " & txtPaciente.Text & " - Diagnóstico: " & txtDiagnostico.Text)
    txtPaciente.Clear()
    txtDiagnostico.Clear()
End Sub
```

Programa 286. Sistema médico con facturación
Controles:

- TextBox → Name: `txtPaciente`
- TextBox → Name: `txtMonto`
- Button → Name: `btnFacturar`, Text: "Generar Factura"
- ListBox → Name: `lstFacturas`

Código:

```
Private Sub btnFacturar_Click(sender As Object, e As EventArgs) Handles btnFacturar.Click
    lstFacturas.Items.Add("Paciente: " & txtPaciente.Text & " - Monto: $" & txtMonto.Text)
    txtPaciente.Clear()
    txtMonto.Clear()
End Sub
```

Programa 287. Sistema médico con reportes
Controles:

- Button → Name: `btnReporte`, Text: "Generar Reporte"
- ListBox → Name: `lstHistorial`

Código:

```
Private Sub btnReporte_Click(sender As Object, e As EventArgs) Handles
btnReporte.Click
    Dim reporte As String = "Reporte de Historial Clínico:" &
Environment.NewLine
    For Each item In lstHistorial.Items
        reporte &= item.ToString() & Environment.NewLine
    Next
    MessageBox.Show(reporte)
End Sub
```

Programa 288. Sistema médico con gráficos
Controles:

- Chart → Name: `chartCitas`, ChartType: Column
- Button → Name: `btnGraficar`, Text: "Mostrar Gráfico"
- ListBox → Name: `lstCitas`

Código conceptual:

```
Private Sub btnGraficar_Click(sender As Object, e As EventArgs) Handles
btnGraficar.Click
    chartCitas.Series.Clear()
    Dim serie As New DataVisualization.Charting.Series("Citas")
    serie.ChartType = DataVisualization.Charting.SeriesChartType.Column
    For Each item In lstCitas.Items
        serie.Points.AddXY(item.ToString(), 1)
    Next
    chartCitas.Series.Add(serie)
End Sub
```

Programa 289. Sistema médico con exportación Excel
Controles:

- Button → Name: `btnExportar`, Text: "Exportar a Excel"

Código conceptual:

```
Imports Microsoft.Office.Interop.Excel


Private Sub btnExportar_Click(sender As Object, e As EventArgs) Handles
btnExportar.Click

    MessageBox.Show("Datos exportados a Excel (requiere referencia a Microsoft
Excel)")

End Sub
```

Programa 290. Sistema médico con importación Excel
Controles:

- Button → Name: `btnImportar`, Text: "Importar desde Excel"

Código conceptual:

```
Imports Microsoft.Office.Interop.Excel


Private Sub btnImportar_Click(sender As Object, e As EventArgs) Handles
btnImportar.Click

    MessageBox.Show("Datos importados desde Excel (requiere referencia a
Microsoft Excel)")

End Sub
```

Programa 291. Sistema médico con múltiples usuarios
Controles:

- ComboBox → Name: `cmbUsuarios` (Items: "Admin", "Doctor", "Recepción")
- Button → Name: `btnAcceder`, Text: "Acceder"

Código:

```
Private Sub btnAcceder_Click(sender As Object, e As EventArgs) Handles
btnAcceder.Click

    MessageBox.Show("Usuario: " & cmbUsuarios.SelectedItem & " ha iniciado
sesión")

End Sub
```

Programa 292. Sistema médico con permisos
Controles:

- ComboBox → Name: `cmbRoles` (Items: "Admin", "Doctor")
- Button → Name: `btnEliminarRegistro`, Text: "Eliminar Registro"

Código:

```
Private Sub btnEliminarRegistro_Click(sender As Object, e As EventArgs) Handles btnEliminarRegistro.Click
    If cmbRoles.SelectedItem = "Admin" Then
        MessageBox.Show("Registro eliminado")
    Else
        MessageBox.Show("No tiene permisos para eliminar")
    End If
End Sub
```

Programa 293. Sistema médico con auditoría
Controles:

- TextBox → Name: `txtPaciente`
- Button → Name: `btnAgregarPaciente`, Text: "Agregar Paciente"
- ListBox → Name: `lstAuditoria`

Código:

```
Private Sub btnAgregarPaciente_Click(sender As Object, e As EventArgs) Handles btnAgregarPaciente.Click
    lstAuditoria.Items.Add(DateTime.Now.ToString() & " - Paciente agregado: " & txtPaciente.Text)
    txtPaciente.Clear()
End Sub
```

Programa 294. Sistema médico con conexión remota
Controles:

- Button → Name: `btnConectar`, Text: "Conectar a Servidor"

Código conceptual:

```
Private Sub btnConectar_Click(sender As Object, e As EventArgs) Handles btnConectar.Click
    MessageBox.Show("Conexión remota establecida (simulada)")
End Sub
```

Programa 295. Sistema médico con backup automático
Controles:

- Button → Name: `btnBackup`, Text: "Realizar Backup"

Código conceptual:

```
Private Sub btnBackup_Click(sender As Object, e As EventArgs) Handles
btnBackup.Click

    MessageBox.Show("Backup automático de datos médicos realizado
correctamente")

End Sub
```

Programa 296. Sistema médico con restauración de datos
Controles:

- Button → Name: `btnRestaurar`, Text: "Restaurar Backup"

Código conceptual:

```
Private Sub btnRestaurar_Click(sender As Object, e As EventArgs) Handles
btnRestaurar.Click

    MessageBox.Show("Datos médicos restaurados desde backup")

End Sub
```

Programa 297. Sistema médico con notificaciones
Controles:

- NotifyIcon → Name: `NotifyIcon1`, Icon: `icono.ico`
- Button → Name: `btnNotificar`, Text: "Notificar Evento"

Código:

```
Private Sub btnNotificar_Click(sender As Object, e As EventArgs) Handles
btnNotificar.Click

    NotifyIcon1.Visible = True

    NotifyIcon1.ShowBalloonTip(3000, "Sistema Médico", "Nuevo evento
registrado", ToolTipIcon.Info)

End Sub
```

Programa 298. Sistema médico con recordatorios de citas
Controles:

- DateTimePicker → Name: `dtpCita`
- Button → Name: `btnRecordatorio`, Text: "Recordar Cita"

Código:

```
Private Sub btnRecordatorio_Click(sender As Object, e As EventArgs) Handles
btnRecordatorio.Click

    MessageBox.Show("Recordatorio: Cita programada para el " &
dtpCita.Value.ToShortDateString())

End Sub
```

Programa 299. Sistema médico con estadísticas
Controles:

- ListBox → Name: `lstPacientes`
- Button → Name: `btnEstadisticas`, Text: "Mostrar Estadísticas"

Código:

```
Private Sub btnEstadisticas_Click(sender As Object, e As EventArgs) Handles
btnEstadisticas.Click

    Dim totalPacientes As Integer = lstPacientes.Items.Count

    MessageBox.Show("Total de pacientes registrados: " & totalPacientes)

End Sub
```

Programa 300. Sistema médico con login de usuarios
Controles:

- TextBox → Name: `txtUsuario`
- TextBox → Name: `txtContraseña`, PasswordChar: "*"
- Button → Name: `btnLogin`, Text: "Login"

Código:

```
Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click

    If txtUsuario.Text = "admin" And txtContraseña.Text = "1234" Then

        MessageBox.Show("Login exitoso")

    Else

        MessageBox.Show("Usuario o contraseña incorrectos")

    End If

End Sub
```

Epílogo

Con la finalización de este recorrido a través de los 300 ejercicios de **Visual Basic .NET con formularios**, hemos llegado al cierre de un camino de práctica, aprendizaje y reflexión. Este libro no pretende ser un final, sino un punto de partida hacia proyectos más ambiciosos y personales, donde cada lector pueda aplicar lo aprendido para resolver problemas reales y dar forma a sus propias ideas.

Cada ejercicio aquí planteado ha tenido un propósito: reforzar conceptos, mostrar técnicas, abrir puertas a nuevas formas de pensar y, sobre todo, transmitir que la programación no es un arte reservado a unos pocos, sino una herramienta accesible para todo aquel que decida dedicarle tiempo, esfuerzo y constancia.

Al concluir esta obra, mi mayor satisfacción es saber que los lectores cuentan ahora con una base más sólida para continuar su desarrollo en este maravilloso mundo. La programación en VB.NET, con toda su potencia y versatilidad, puede ser tanto un medio de formación profesional como una vía de expresión personal, y el dominio adquirido en estos ejercicios es el primer paso hacia esa libertad creativa.

Deseo que lo aprendido en estas páginas no se quede solo en la teoría, sino que inspire a experimentar, a investigar más allá de lo escrito y a no temer equivocarse. Porque en cada error hay una oportunidad de mejorar, y en cada programa terminado hay un logro que nos impulsa a seguir.

Que este libro sirva como una base, un compañero de aprendizaje y, sobre todo, como una motivación constante para seguir avanzando. El camino de la programación es largo, pero también apasionante, y estoy seguro de que quienes lo recorren con paciencia y entusiasmo logran alcanzar metas que parecían inalcanzables.

Este es solo un cierre en papel, pero para ti, lector, espero que sea un **nuevo comienzo**.